BORLAND LANGUAGE EXPRESS

VOLUME 1 • NUMBER 4 WINTER 1992

Now over 500,000 readers and still growing strong.

LATE NEWS

Enter Borland's "Spring-Along-The-Seine" Sweepstakes

We know you'd love a week in Paris, so we made it easy for you to qualify for our "Spring-Along-the-Seine" Sweepstakes!

First Prize is a fabulous week in Paris for two! Think of all that culture and adventure—plus the chance to bump into renowned French programmers along the Champs Elysee!

Second Prize winners get their choice of any five Borland products. While Third Prize winners can select any Borland language product.

Just check the appropriate box on your FREE Issue Card and you'll be entered. It's that easy. Do it now!

SCOTTS VALLEY, CA — The headline is not a misprint. In less than a year, the Borland Language Express has grown from an exciting idea into an informative and dynamic publication with over 500,000 readers worldwide.

What has amazed our editors is that as its readership grows, the Borland Language Express is attracting more and more world-class programmers anxious to share their knowledge. Every day we receive a new batch of proposals for articles. Deciding which articles to publish has been a challenge.

As a result, if you liked 1991, you're going to love 1992. We can promise you an exceptional editorial calendar for the year. We have things that every programmer is sure to enjoy, from novice to expert. For instance, a continuing series of articles covering Turbo Pascal, C, C++ and other languages. The emphasis will continue to be on education, with complete step-by-step instructions to let you easily use these new ideas in your own programming. You definitely won't want to miss a word.

That's why it's important to send in your FREE Issue Card now. All it takes is a few minutes — and it will guarantee that you will receive the next issue of Borland Language Express and continue to be one of the programmers in the know!

World's leading programmers to converge on Monterey

At press time, we continue to receive cables, faxes and mail deliveries of registration forms for the **Borland International Developers Conference** to be held April 12-15, 1992 in Monterey, California.

BEST CONFERENCE YET. The 1990s promises to be the "Decade of the Programmer." And the Monterey Conference will offer you a rare opportunity to hear about all the latest in programming, right from the sources!

Question some of the world's greatest programmers. Hear celebrated keynote speakers, and attend a wide range of user group meetings, tutorials, discussion groups, informal meetings and more.

FREE GIFT FOR ATTENDEES. Once your Conference registration is received and paid for, you qualify for an extra treat — your choice of any **Borland language product free.** You name it, it's yours! *Register now* and we'll ship your gift to you immediately.

So hurry. The registration deadline ends soon. To reserve your space or for more details, call **1-800-942-8872.** The remaining space is extremely limited.

Borland Language Express FREE Issue Form

❑ YES! Add my name to continue receiving complimentary issues of Borland Language Express!

❑ YES! I'm very interested in the '92 Borland International Developers Conference.

❑ YES! Enter my name in the Borland Language Express Sweepstakes!

ZAE03

Your input is important to us. To continue receiving your free issue of Borland Language Express, we ask that you please take a few minutes to complete the form below. (All questions must be completed.) Thank you for your comments.

COMPANY INFORMATION (Check all that apply):

1. Which operating system(s) are you currently using:
❑ DOS ❑ Windows ❑ Mac ❑ OS/2 ❑ UNIX
2. Which operating system(s) are you planning to use in the future:
❑ DOS ❑ Windows ❑ Mac ❑ OS/2 ❑ UNIX
3. Number of employees at your site:
❑ 0-50 ❑ 50-100 ❑ 100-500 ❑ Over 1000
4. Which Borland Language product(s) are you currently using:
❑ Turbo C++ ❑ Turbo C++ for Windows
❑ Turbo Pascal ❑ Turbo Pascal for Windows
❑ Borland C++ ❑ Borland C++ & Application Frameworks
❑ ObjectVision ❑ Video training courses
❑ Other Borland Products — Please specify:

Daytime Phone Number: (   ) ____________________

❑ If this is not your correct name and address, please check here and make necessary corrections above.

P/N BK-LAN008

Dear Reader:

DON'T LET THIS BE YOUR LAST ISSUE!

To assure that you'll continue receiving the Borland Language Express FREE, please complete and return the survey card on the front of this issue.

While you're at it, don't forget to enter Borland's exciting "*Spring-Along-the-Seine*" Sweepstakes. You could win an incredible week for two in Paris or free Borland products.

Also be sure to check the box for information on the *Borland International Developers Conference* to be held this April in Monterey, California. It's an event that no serious programmer will want to miss.

So don't risk losing out on the latest programming developments — or missing out on the Sweepstakes. Just complete and return the survey card *now!*

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 200 SANTA CRUZ, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

BORLAND

TECHNICAL DEVELOPMENT CENTER
1800 Green Hills Road
P O Box 660005
Scotts Valley CA 95067-9985

BORLAND LANGUAGE EXPRESS

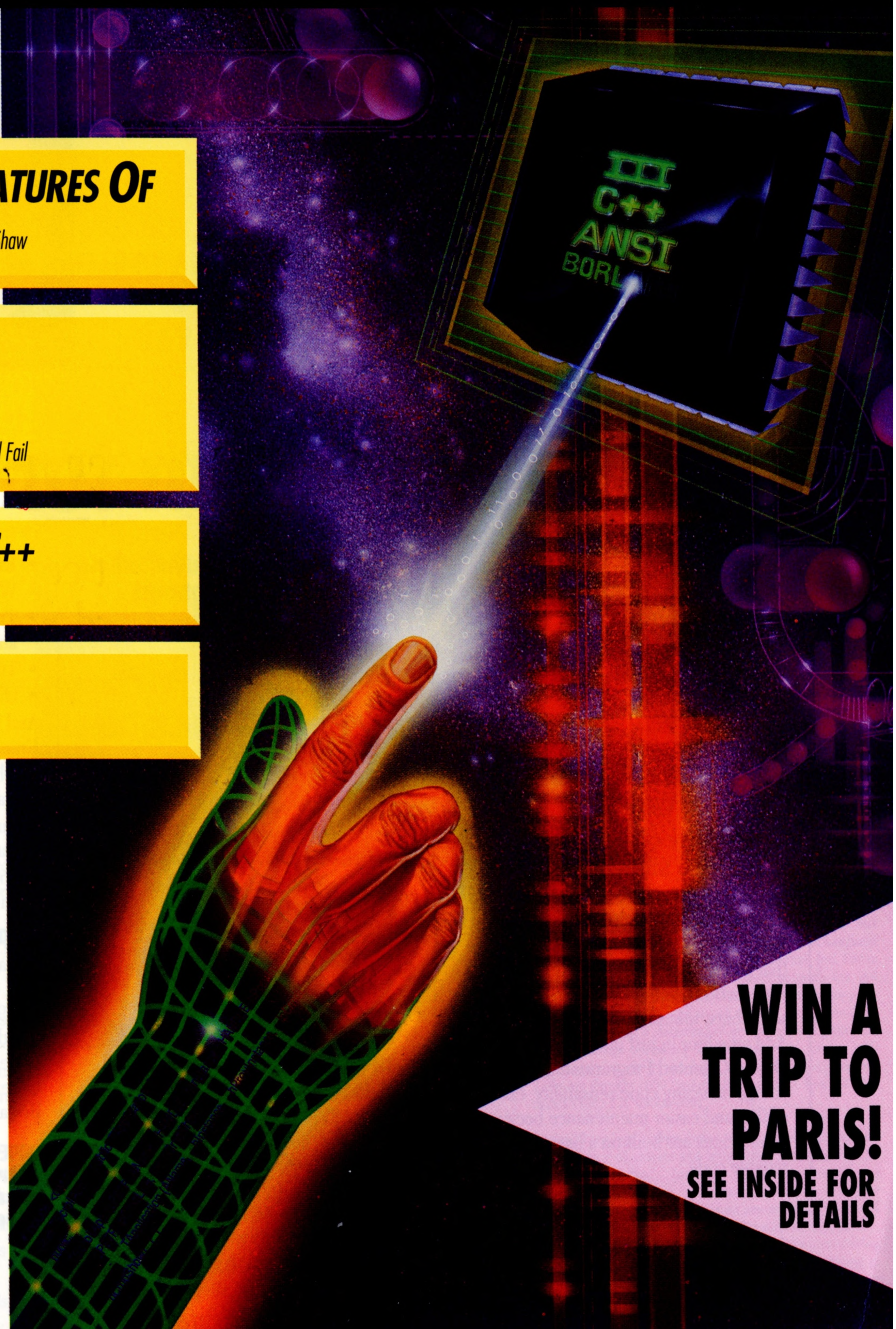
WINTER 1992 US $10
VOLUME 1 NUMBER 4

The Optimization Features Of Borland C++
By Richard Hale Shaw

ObjectVision As An Object-Oriented Development Tool
By David Fail

Templates In Borland C++
By Bruce Eckel

Turbo Technix®
Answers to Your Questions



WIN A TRIP TO PARIS!
SEE INSIDE FOR DETAILS

TIMELY INFORMATION FOR TODAY'S PROGRAMMER

NEW! VERSION 3.0

...If you program for a living

New! Borland® C++ & Application Frameworks 3.0 is *the* choice of professional C and C++ programmers for Windows and DOS application development. With unmatched optimizations, powerful tools, unsurpassed Windows development environment and object-oriented application frameworks, Borland C++ & Application Frameworks 3.0 has no equal. Quite simply, if you program for a living, this is everything you need.

OOP, to simplify your life

Borland C++ & Application Frameworks 3.0 simplifies programming by giving you ready-made user interface objects that plug directly into your application. Automatically inherit windows, menus, scroll bars, mouse support and more. Add an editor in just one line. With Object-Oriented Programming (OOP), you get amazing code reusability, extensibility and easier maintenance because applications are built on a base of tested, reliable code.

New features give you incredible programming options!

Just look at some of the enhanced features in Borland C++ 3.0:

- ANSI C and C++ 2.1 and templates
- Global optimizer includes:
  - Global register allocation
  - Local and global common sub-expressions
  - Induction variables
  - Loop and jump optimization
  - Register parameter passing
  - And ten other state-of-the-art optimizations
- Increased C++ compile speed
- Windows and DOS Integrated Development Environments
- Visual ObjectBrowser™ to view class relationships at a glance
- DPMI support for compiler and IDE environments gives you huge capacity
- EasyWin™ library makes it easy to convert your DOS programs to Windows
- Resource Workshop for creating Windows user interfaces visually
- Extensive Microsoft® C compatibility
- WinSight™ message tracking utility
- Turbo Debugger® for DOS and Windows
- Turbo Profiler™ for DOS and Windows
- Object-oriented Turbo Assembler®

And with new Borland C++ & Application Frameworks 3.0 you get all of this, plus:

- ObjectWindows™—the application framework for Windows
- Turbo Vision™—the application framework for DOS
- Source code for runtime library and application frameworks

Optimized for professionals

Borland C++ 3.0 ($495.00*) or Borland C++ & Application Frameworks 3.0 ($749.00*) are optimized for your life-style. But don't wait. Because when it comes to professional programming, there's no better way to earn a living than with Borland C++.

ORDER NOW!
To order, use the attached order form or call 1-800-331-0877 or see your retail dealer.

B O R L A N D

The Leader in Object-Oriented Programming

a word from the EDITOR

DAVID INTERSIMONE

HAPPY NEW YEAR FROM BORLAND!

I spent a great holiday vacation, playing with the new C++ compilers and visual programming tools that we shipped in November. I hope that you were also rewarded with optimizing compilers, application frameworks, visual programming and windows tools. But if a grinch visited your house, don't worry; there is still time to get your hands on the latest versions.

We are now into what Philippe Kahn has called the "Decade of Software." Programmers are taking advantage of object-oriented programming languages, the growing number of class libraries, visual programming, and state-of-the-art browsing and development tools to develop a new generation of powerful and easy-to-use software. As a result, they are getting a jump start to meet the growing demands of computer users and to respond to rapid changes in hardware and operating systems software.

It seems like only yesterday (it was almost 3 years ago, in fact) that Turbo C++ 1.0 was released. Now, Borland C++ 3.0 is our third-generation C++ compiler, bringing the latest additions to the ANSI C++ language along with a fast professional-strength global optimizer. Richard Hale Shaw probes the new global optimizer, explaining many of the optimizations and quantifying the speed and size gains achievable, in "Generating the Smallest, Fastest Code: The Optimization Features of Borland C++ 3.0." Bruce Eckel's "Templates in Borland C++" teaches you how to use templates, a new C++ language capability that provides even more type safety.

Visual programming with ObjectVision enables every computer user to take advantage of his or her domain of expertise to build robust Windows applications without programming. David Fail, with "ObjectVision 2.0 as an Object-Oriented Development Tool," shows how to use ObjectVision to design and develop custom applications and how to add new functionality with Dynamic Link Libraries (DLLs).

If it seems like the world of software development is accelerating beyond your ability to assimilate it all, don't worry. The Second Annual Borland International Developers Conference happens in Monterey, California, April 12-15. Over a thousand developers from around the world will learn the latest in OOP, C++, Pascal, visual programming, operating systems and application design from industry-leading developers and Borland's own development staff. During four days of concentrated learning and technical fun, attendees will become experts in the latest development technologies.

Our regular *Turbo Technix* section has the answers to the most common questions that our technical support department has received since the last *Borland Language Express*.

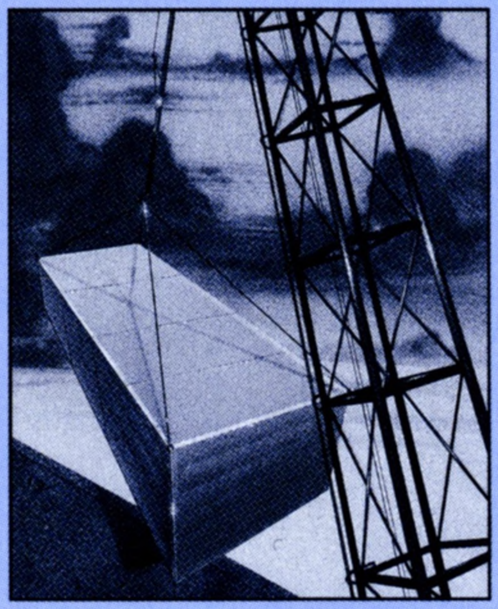
It's time to power into the new year, improving your skills and harnessing the latest technologies and tools. I just know it is going to be a great year for all developers!

CONTENTS

FEATURE

Page 4

ARTICLES



Page 16

DEPARTMENTS

Page 22

PUBLISHER
Brian Anderson

EDITOR
David Intersimone

CONTRIBUTING EDITORS
Richard Hale Shaw, Peter Becker, Bruce Eckel, David Fail, Christine Sherman

TECHNICAL REVIEWERS
Peter Becker, Kathleen Garvey, Daniel Horn, Spencer Kimball, Peter Kukol, Phil Rose, Jeff Stock, Zack Urlocker

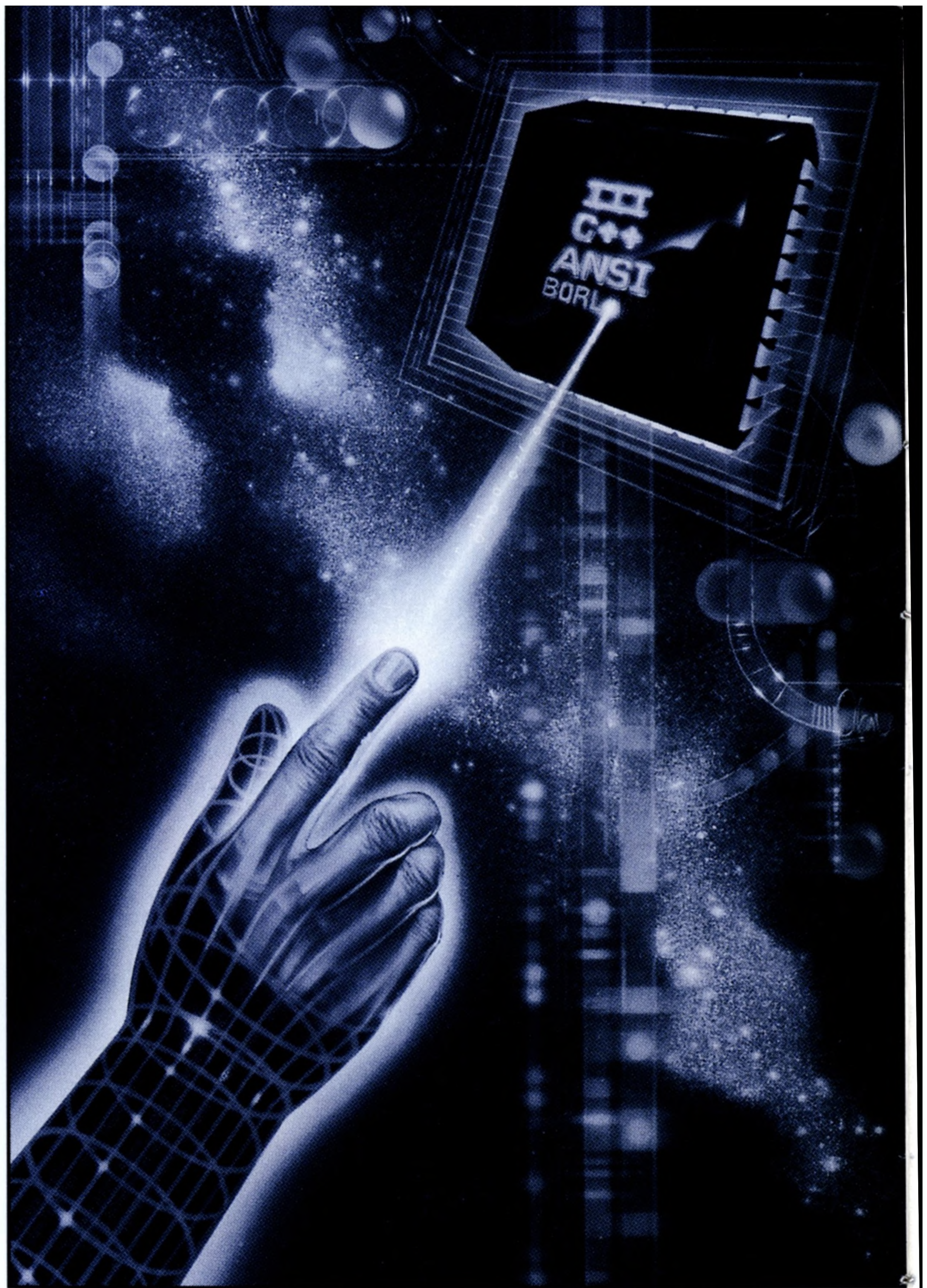
PRODUCTION MANAGER
Janel Killheffer

PRODUCED BY
Apple Marketing, Inc.
Los Angeles, CA

Borland Language Express *is printed and manufactured in the U.S.A.*

GENERATING THE SMALLEST, FASTEST CODE:

The Optimization Features of Borland C++ 3.0



BY RICHARD HALE SHAW

Ever since the introduction of Turbo C 1.0, developers have asked Borland for one potentially powerful feature: optimization facilities. With optimization, a compiler not only translates your C code into executable form, but it analyzes the resulting assembly language constructs, and massages and rearranges them into the fastest-executing, smallest code that it can. With the release of Borland C++ 3.0, you can now optimize your applications to produce even smaller, faster programs than you could with Borland C++ 2.0.

This article examines the efficacy of the optimizer found in Borland C++ 3.0's code generator. Despite the fact that this is Borland's first optimizing compiler, it's surprisingly mature. And while I expect that it will become even more powerful over time, I've found that its generated code comes surprisingly close to — and in some cases equals — the speed and size of code generated by Microsoft C 6.0.

In this article, I'll discuss several optimization techniques used by optimizing compilers. I'll demonstrate each one via an example in C, and show you the effect of that optimization if you'd implemented it in C, by hand. Then we'll look at the resulting assembly language when the C code is run through Borland C++ 2.0, Microsoft C 6.0 and Borland C++ 3.0.

I'll assume that you know enough assembler to read the fragments of assembly language shown here. (You'll only need to be slightly familiar with a handful of assembly language instructions: MOV, PUSH, POP, INC, ADD, SUB, XOR, CMP, RET, NOP, SHL, REP STOSW and some of the JMP variations.)

How the Optimization Tests were Performed

For each optimization test, I wrote at least two functions of the C code to be optimized. The functions represent the two versions of the code (the plain, vanilla version and the hand-optimized version). Using these two functions, testfunc1 and testfunc2, you can test two versions of functionally identical code under identical conditions: each function has the same calling sequence and is compiled in the same module under the same set of compilation conditions.

I compiled these functions with each of the three compilers, turning on all optimization features, with a preference for execution speed over code size. Then, to time the execution of each function, I linked them into TIMER.EXE, a timing program that provides a framework for timing each function as it's called over and over from a loop.

After repeated tests, I've found that TIMER yields some amazingly consistent results, which are shown in the table titled OPTIMIZATION BENCHMARK RESULTS. Let's take a closer look at several of the optimization tests, how Borland C++ 3.0 compared to Borland C++ 2.0 and Microsoft C 6.0 — and why.

Examining the Benchmark Results

The Strength Reduction Test

Strength reduction is an optimization technique for reducing or eliminating the overhead (or "strength") of code in a program loop. For example, consider the following loop:

```
#include"timer.h"
// defines NUMELEMENTS
char values[100];
void testfunc1(void) {
  int counter;
  for(counter = 0;
      counter <
        NUMELEMENTS(values);
      counter++)
    values[counter] = '\0';
}
```

To apply strength reduction here, you'd attempt to reduce the loop's strength: specifically, the number of loop iterations and the setting of the 'values' array to zero.

A non-optimizing compiler, Borland C++ 2.0 generates the following assembler from this C code:

```
TESTFUNC1   proc   near
  push   si
  xor si,si
  jmp short @2@98
@2@50:
  mov byte ptr DGROUP:VALUES[si],0
  inc si
@2@98:
  cmp si,100
```

```
  jl short @2@50
  pop si
  ret TESTFUNC1
  endp
```

This is largely an assembly language translation of the C code: the 'counter' variable is tracked in the SI register and is used to calculate the array offsets and when the loop should terminate. On the test machine I used, in a loop of 1,000,000 iterations, this code runs in 53 seconds.

An optimizing compiler will attempt to find a faster way. For example, here's the result of compiling this code with Microsoft C:

```
@testfunc1  PROC NEAR
  push   di
  sub ax,ax
  mov di,OFFSET DGROUP:_values
  mov cx,50
  push   ds
  pop es
  rep stosw
  pop di
  ret
  nop
@testfunc1  ENDP
```

This code uses a single instruction, REP STOSW to initialize the array, 'values'. The STOSW instruction can copy the 16-bit word from AX into the location pointed to by ES:DI. By prefixing it with REP, the STOSW will repeat for the number of iterations specified in the CX register. Thus, a loop is still in effect, but with half as many iterations: since STOSW uses 16-bit words (instead of 8-bit bytes) it's possible to cut the number of loop iterations in half. The loop counting and address offset calculations are automatically performed by this single instruction.

Using Borland C++ 3.0 on this example, you can get the same optimization benefits as with Microsoft C:

```
_testfunc1  proc    near
  push    di
  mov cx,50
  mov di,offset DGROUP:_values
  push    ds
  pop es
  xor ax,ax
  rep     stosw
  pop di
  ret
_testfunc1  endp
```

Note that the techniques used are identical to those in the Microsoft C example, but the order of events is slightly different: i.e., CX is initialized first, then DI, then ES is setup, and finally, AX is cleared. Also, note that AX is cleared via an XOR, while Microsoft C uses a SUB instruction. Consequently, when this version is compiled with Borland C++ 3.0, it runs in the same amount of time as did the Microsoft C6-generated code, and in less than one-fifth the time required by the BC++ 2.0 code: about 10 seconds.

Hand-Optimized Strength Reduction

Now suppose that you didn't have an optimizing compiler, and wanted to re-write this function in an attempt to optimize it by hand:

```
char values[100];
void testfunc2(void) {
  register int counter = 0;
  register char *p = values;
  for( ;
      counter <
        NUMELEMENTS(values);
      counter++, p++)
    *p = '\0';
}
```

Here, testfunc2 uses a pointer to the 'values' array and keeps both the pointer and the counter in register while it increments them.

Both the BC++ 2.0 and 3.0 versions of this function perform at about the same speed (59 seconds and 56 seconds, respectively) — about 5 times slower than the code generated by Microsoft C.

Note that you might have been smart enough to simply use the memset function to initialize the array, and remove the loop altogether:

```
memset(values,0,
  NUMELEMENTS(values));
```

If you did, and used either version of Borland C++, you're going to get better performance: the 3.0 compiler replaces the code with a REP STOSW and runs at about 10 microseconds per iteration. The 2.0 compiler calls memset (which also uses REP STOSW), but combined with the function call overhead, requires 14 microseconds per iteration.

Surprisingly, the Microsoft compiler also replaces the memset call — but with a REP STOSB. This instruction only copies one 8-bit byte at a time, and is consequently slower than the combination of the function call to memset and REP STOSW used by BC++ 2.0. Thus, the Microsoft version runs at over 19 microseconds per iteration.

Further Strength Reduction Tests

Strength reduction isn't always accomplished as easily as in the examples above. In those examples, we initialized an array to zero: what if a program

ANSI C++ (X3J16) COMMITTEE NEWS

by Peter Becker *(Borland committee member)*

The biggest news about the ANSI C++ committee is that it is now the ANSI/ISO C++ committee. For those of you who aren't into bureaucratic abbreviations, ANSI is the American National Standards Institute, and ISO is the International Standards Organization. So the new committee name means that one committee is developing both the American standard and the International standard for C++.

The ANSI committee has always tried to keep international issues in mind during its deliberations, but it's easy for us in the United States to forget that the way we do things isn't necessarily the way that the rest of the world does things. This change into an international committee will help widen that parochial view, both because more international representatives will take part in the deliberations that lead up to the standard, and because the standard itself will be subject to formal review worldwide.

The following are some of the problems that have been mentioned in committee discussions. It is by no means exhaustive, and is intended only to suggest the sorts of problems that an international language standard has to deal with:

1) keyboards that don't have all the ASCII characters (no {, }, |, |, etc.)
2) representation of characters used in languages other than English
3) alphabetizing strings in various language representations
4) input and output of character strings.

One of the big questions that the ANSI/ISO committee has to resolve is whether the solutions to these problems contained in the C standard are adequate, and if not, what a more complete solution should look like. There is a great deal of work being done by people outside the ANSI/ISO committee to develop workable international standards for national language support in programming languages, and the committee needs to take a close look at that work and decide how much of it, if any, to incorporate into the C++ language standard.

loop is initializing an array with different values altogether? For example:

```
int values[100];
void testfunc1(void) {
  int counter;
  for( counter = 0;
       counter < 100;
       counter++)
    values[counter] = counter * 8;
}
```

The elements of the 'values' array are set to the current loop counter value multiplied by 8. Again, BC++ 2.0 generates code that implements what's described in the loop code in C:

```
TESTFUNC1   proc    near
  push    si
  xor si,si
  jmp short @2@98
@2@50:
  mov ax,si
  shl ax,1
  shl ax,1
  shl ax,1
  mov bx,si
  shl bx,1
  mov word ptr
    DGROUP:VALUES[bx],ax
  inc si
@2@98:
  cmp si,100
  jl  short @2@50
  pop si
  ret
TESTFUNC1   endp
```

Here, SI is the loop counter and its value is shifted left three times to perform the multiple by 8. BX is used as an offset into the array, by taking the counter value and shifting it left once (thus doubling it so it can be used as a 16-bit offset, rather than an 8-bit one). This code takes 95 seconds to run in the test loop.

Microsoft C 6.0 offers a more efficient approach:

```
@testfunc1  PROC NEAR
  sub dx,dx
  mov bx,OFFSET DGROUP:_values
$F165:
  inc bx
  inc bx
  mov WORD PTR [bx-2],dx
  add dx,8
  cmp bx,OFFSET
    DGROUP:_values+200
  jb  $F165
  ret
@testfunc1  ENDP
```

With this approach, the role of loop counter is eliminated: DX contains the array value and BX is the array offset. The address of the end of the array is used for loop termination instead of a loop counter. BX is incremented with every loop iteration, and the value in DX is loaded into BX-2. Then 8 is added to DX and the address in BX is compared to the end of the array. This takes two-thirds of the time required by the BC++ 2.0 code, and clocks in at 64 seconds during the test.

Borland C++ 3.0 uses a similar, but slightly faster approach:

```
_testfunc1  proc    near
  push    si
  xor dx,dx
  mov si,offset DGROUP:_values
@2@58:
  mov word ptr [si],dx
  add dx,8
  add si,2
  cmp dx,800
  jne short @2@58
  pop si
  ret
_testfunc1  endp
```

DX is still the array value, but SI is the array offset. Rather than incrementing it beforehand and using SI-2, SI is incremented afterward via an ADD rather than two INCs. And rather than comparing the array offset to the end of the array for loop termination, the array offset is used — like a loop counter — instead; the code compares DX to its ultimate value, 800. Consequently, the BC++ 3.0 code consistently executes this code about 3 microseconds faster per iteration than does the Microsoft C code — and runs 3 seconds faster during the test, at 61 seconds.

Loop Unrolling

Loop unrolling is another compiler optimization technique. It attempts to make loops more efficient by reducing the number of loop iterations at the expense of generating more code. Take, for example, a loop like the following:

```
int z[100], y[100];
void testfunc1(void) {
  int counter;
  for(counter = 0;
      counter < NUMELEMENTS(z);
      counter++)
    z[counter] = (y[counter] * 8);
}
```

Each iteration initializes a member of the 'z' array with a value drawn from the 'y' array. You can unroll the loop by performing more than one initialization with each iteration — and thus cut in half the number of iterations. The effect would be the same as in the following code:

```
void testfunc2(void) {
  int counter;
  for(counter = 0;
      counter < NUMELEMENTS(z);
      counter += 2) {
    z[counter] = (y[counter] * 8);
    z[counter + 1] =
      (y[counter + 1] * 8);
  }
}
```

Note that — theoretically, at least — you can unroll a loop completely by writing out each iteration and removing the loop altogether. With loop unrolling, you trade off a code space (here, the space for the extra code during each loop iteration) and pick up speed by reducing the total number of iterations.

To test the effectiveness of this technique, I compiled both of these pieces of code with each of the three compilers. Borland C++ 2.0 clocked testfunc1 at 117 seconds with the following code:

```
TESTFUNC1   proc    near
  push    si
  xor si,si
  jmp short @2@98
@2@50:
  mov bx,si
  shl bx,1
  mov ax,word ptr DGROUP:Y[bx]
  shl ax,1
  shl ax,1
  shl ax,1
  mov bx,si
  shl bx,1
  mov word ptr DGROUP:Z[bx],ax
  inc si
@2@98:
  cmp si,100
  jl  short @2@50
  pop si
  ret
TESTFUNC1   endp
```

Note that this function uses SI as the loop counter, and unnecessarily reloads BX with SI twice during each loop iteration. It would have been easier to load BX once. Plus, rather than loading AX and shifting it three times, why not change the value in AX with a single shift instruction?

As you can see below, Microsoft C makes pretty neat work of it:

```
@testfunc1  PROC NEAR
  sub bx,bx
$F166:
  mov cl,3
  mov ax,WORD PTR _y[bx]
```

```
    shl ax,cl
    mov WORD PTR _z[bx],ax
    inc bx
    inc bx
    cmp bx,200
    jl $F166
    ret
    nop
  @testfunc1 ENDP
```

This code loads CL with the number of times AX will be shifted (3), loads AX with the value from the 'y' array, and then shifts it to perform the multiplication. It then immediately stores that value in the 'z' array. All that remains is to increment BX (the loop counter), and check for loop termination by comparing it to 200 (since these are arrays of 100 16-bit words, the loop counter is compared with 200, not 100). Clocking in at 84 seconds, this Microsoft C code runs about 37% faster than the BC++ 2.0 version. However, note that while Microsoft C generates very efficient code for this function, it does not unroll the loop.

What about BC++ 3.0? It generates the following code:

```
  _testfunc1 proc    near
    push   si
    xor si,si
  @2@58:
    mov ax,word ptr DGROUP:_y[si]
    shl ax,1
    shl ax,1
    shl ax,1
    mov word ptr DGROUP:_z[si],ax
    add si,2
    cmp si,200
    jne short @2@58
    pop si
    ret
  _testfunc1 endp
```

What are the differences? First, SI is preserved with a PUSH and POP at the beginning and end of the function. It uses SI as the array offset variable (MC 6.0 uses BX instead). And while the function uses AX in the same way as the MC 6.0 code, it still performs the multiple shifts (where one would do). So while it's more efficient than BC++ 2.0 (89 seconds to BC++ 2.0's 117), it's a little slower than the code generated by MC 6.0. Plus, it still doesn't unroll the loop.

Now, being the knowledgeable, talented programmer you are, what if you used the (partially) unrolled version of this loop, shown above in testfunc2? If you do, you'll find that your BC++ 2.0 version will run about 10% faster at 100 seconds. With Borland C++ 3.0, you'll find it also shaves 11 seconds off the performance of testfunc1. However, Microsoft C's approach does even better: it shaves 26 seconds — over 20% — off the performance of an already fast function.

If you're wondering why, the answer is simple: All three compilers simply unroll the loop as the code describes. It's the cumulative effect of the faster code generated by Microsoft C that has a profound impact as the number of loop iterations are reduced. In other words, the more you unroll this loop, the faster its code will run.

The point is: Here's an optimization technique that you're better off applying by hand — especially for crucial, performance-dependent loops that don't have a relatively large amount of code being executed per iteration.

Induction Variable Elimination

This optimization attempts to reduce the strength of a loop by removing loop counters and recursively defined variables. For instance, in the following code,

```
  #include"timer.h"
   // defines NUMELEMENTS
  int values[100];
  void testfunc1(void) {
    int counter;
    for( counter = 0;
         counter <
           NUMELEMENTS(values);
         counter++)
      values[counter] = counter * 8;
  }
```

the loop counter is simultaneously used to generate the numbers that initialize the array and the loop termination value. Note that the highest value to be inserted in the array is (NUMELEMENTS(values) - 1)*8), and that the array will contain a series of numbers, each greater than the previous by 8. Consequently, we can change the loop termination value to (NUMELEMENTS(values)*8), generate the array values by adding 8 to a register through each loop iteration, and use a pointer to generate the array offset addresses:

```
  #include"timer.h"
  // defines NUMELEMENTS
  int values[100];
  void testfunc2(void) {
    register int t = 0;
    register int *p;
    for(p = values, t = 0;
        t <
          NUMELEMENTS(values)*8;
        t += 8, p++)
      *p = t;
  }
```

Here's what Borland C++ 2.0 generates when it encounters testfunc1:

```
  TESTFUNC1   proc   near
    push   si
    xor si,si
    jmp short @2@98
  @2@50:
    mov ax,si
    shl ax,1
    shl ax,1
    shl ax,1
    mov bx,si
    shl bx,1
    mov word ptr
      DGROUP:VALUES[bx],ax
    inc si
  @2@98:
    cmp si,100
    jl  short @2@50
    pop si
    ret
  TESTFUNC1   endp
```

Again, a fairly direct translation of the C-code's original intent. While this runs in 95 seconds, using testfunc2 is even faster:

```
  TESTFUNC2   proc   near
    push   si
    push   di
    xor si,si
    mov di,offset DGROUP:VALUES
    xor si,si
    jmp short @3@98
  @3@50:
    mov word ptr [di],si
    add si,8
    inc di
    inc di
  @3@98:
    cmp si,800
    jl  short @3@50
    pop di
    pop si
    ret
  TESTFUNC2   endp
```

SI is still the loop counter, as in the previous version. But it's incremented by 8 during each iteration, and the loop terminates when SI reaches 800. There are still only 100 iterations, but the array insertion value is always in SI; there's no need to generate it with a bunch of left-shifts (as the previous version does). DI is used as a pointer to each array element, and it's simply incremented with each loop iteration. Consequently, the BC++ 2.0 version of testfunc2 runs in 62 seconds — faster than testfunc1 by a third.

So you're probably wondering how well the two optimizing compilers did with this code. MC 6.0

generated the following from testfunc1:

```
@testfunc1  PROC NEAR
  sub dx,dx
  mov bx,OFFSET DGROUP:_values
 $F165:
  inc bx
  inc bx
  mov WORD PTR [bx-2],dx
  add dx,8
  cmp bx,OFFSET
     DGROUP:_values+200
  jb  $F165
  ret
@testfunc1  ENDP
```

The compiler introduces induction variable elimination; DX carries the array insertion value, while BX is the pointer into the array and is used to determine the loop termination point.

This code runs in 64 seconds — and so does MC's version of testfunc2. The reason? The code is nearly identical:

```
@testfunc2  PROC NEAR
  mov bx,OFFSET DGROUP:_values
  sub dx,dx
 $F171:
  inc bx
  inc bx
  mov WORD PTR [bx-2],dx
  add dx,8
  cmp dx,800
  jl  $F171
  ret
@testfunc2  ENDP
```

Note that only two things changed: the order of the first two lines, and that rather than comparing the address in BX to &values[100] (that's _values+200 in assembler), testfunc2 compares DX to the loop termination value. It's all the same in any case; both versions run in 64 seconds.

Now, here's what Borland C++ 3.0 generates:

```
_testfunc1  proc   near
  push   si
  xor dx,dx
  mov si,offset DGROUP:_values
 @2@58:
  mov word ptr [si],dx
  add dx,8
  add si,2
  cmp dx,800
  jne short @2@58
  pop si
  ret
_testfunc1  endp
```

The Borland compiler uses SI to carry the array offset address, while DX is still used to hold the array insertion value. The real difference is that rather than incrementing the pointer before inserting the array value and then using *ptr-1 (as the MSC-generated code does), this code increments the pointer afterwards, and uses an ADD instead of two INCs.

As with the Microsoft C versions, the BC++ 3.0 version of testfunc2 is nearly identical to its version of testfunc1:

```
_testfunc2  proc   near
  push   si
  mov si,offset DGROUP:_values
  xor dx,dx
 @3@58:
  mov word ptr [si],dx
  add dx,8
  add si,2
  cmp dx,800
  jl  short @3@58
  pop si
  ret
_testfunc2  endp
```

There are only a few subtle differences here. First, the order of initialization of SI and DX are reversed. No big deal. Second, testfunc2 uses a Jump-If-Lower rather than the Jump-If-Not-Equal used by testfunc1.

These differences don't really matter. However, both versions run in 61 seconds — 3 seconds faster than either version generated by Microsoft C 6.0.

Conclusion

These aren't the only optimizations performed by compilers, although they're among the more common ones. But the results of these tests, along with analysis of the assembler output, show what a reliable optimizing compiler can do for the performance of your code — and that there are times when it's worthwhile to introduce some of the optimizations yourself, by hand.

As you can see, the code generated by Borland C++ 3.0 puts it into the same league as that generated by Microsoft C 6.0a. While it still has only a first-generation optimizer, the BC++ 3.0 is surprisingly mature. And frankly, I prefer that Borland proceed carefully: I'd rather have an optimizing compiler that has room for improvement, rather than one that may introduce optimization bugs along the way. But I should add that even with optimization, BC++ 3.0 is still a fast compiler, while MC 6.0 took 64 seconds to compile the test suite, BC++ 3.0 took only 47 seconds.

TABLE 1

OPTIMIZATION BENCHMARK RESULTS

TEST	BC2 ORIGINAL	BC2 HAND-OPT.	MC6 ORIGINAL	MC6 HAND-OPT.	BC3 ORIGINAL	BC3 HAND-OPT.
SR	53	59	10	10	10	56
FSR	95	71	64	64	61	61
LU	117	100	84	58	89	78
IVE	95	62	64	64	61	61

SR=Strength Reduction **FSR=Further Strength Reduction**
LU=Loop Unrolling **IVE=Induction Variable Elimination**

All tests performed on a NorthGate 33mhz 80386-based computer running DOS 5.0.
All times in seconds, rounded to the nearest second.

In any case, the results in Table 1 show how far BC++ has come, and that BC++ developers now have even greater opportunities to write smaller, faster code.

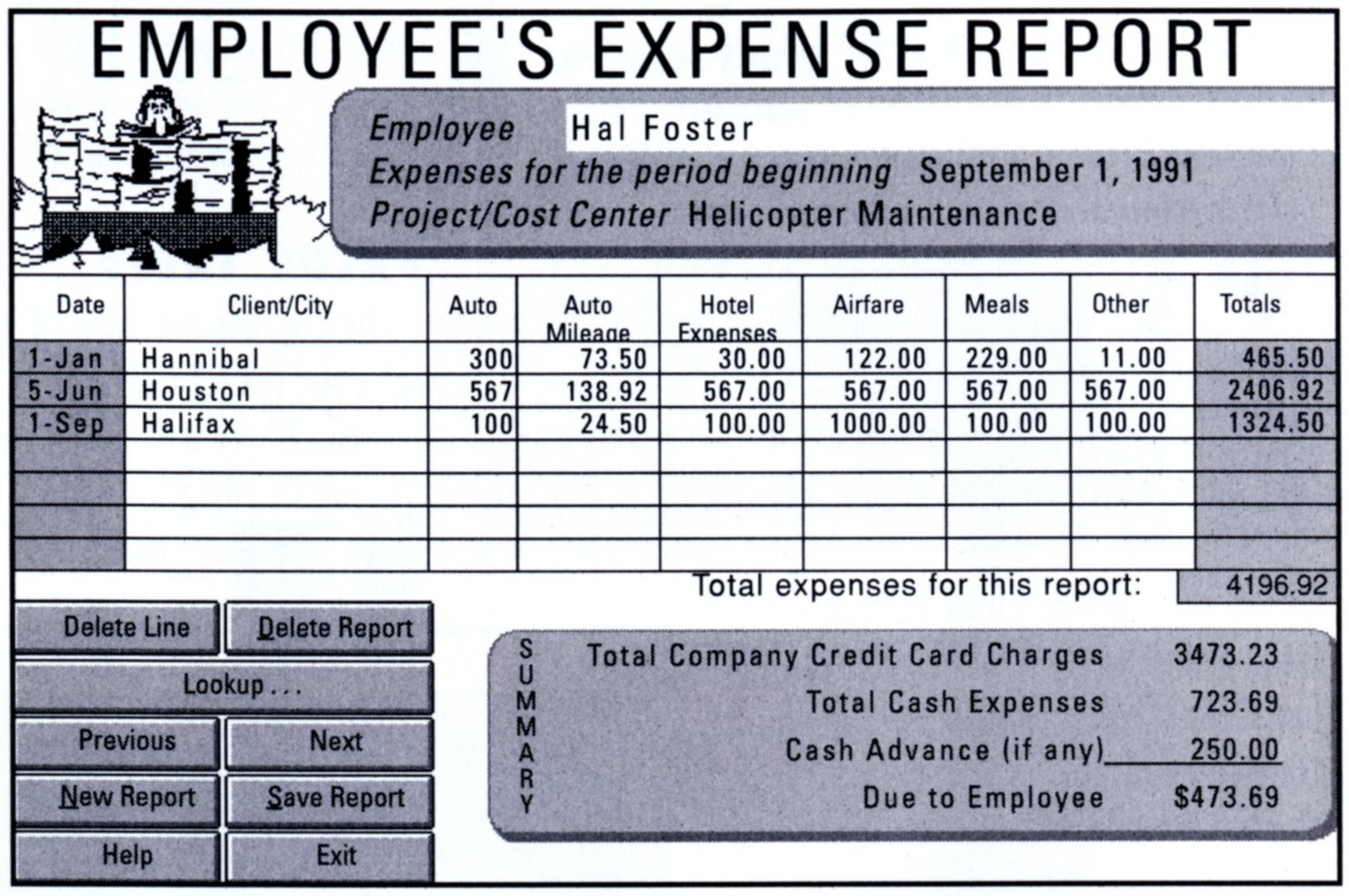
About the Author:

Richard Hale Shaw is a Contributing Editor for PC Magazine *and a frequent contributor to various other computer magazines. He is a member of the Turbo Languages Advisory Board and the author of* C Power Tools, *published by Ziff-Davis Press.*

OBJECTVISION 2.0

AS AN OBJECT-ORIENTED DEVELOPMENT TOOL

BY DAVID FAIL



EMPLOYEE'S EXPENSE REPORT

Employee Hal Foster
Expenses for the period beginning September 1, 1991
Project/Cost Center Helicopter Maintenance

Date	Client/City	Auto	Auto Mileage	Hotel Expenses	Airfare	Meals	Other	Totals
1-Jan	Hannibal	300	73.50	30.00	122.00	229.00	11.00	465.50
5-Jun	Houston	567	138.92	567.00	567.00	567.00	567.00	2406.92
1-Sep	Halifax	100	24.50	100.00	1000.00	100.00	100.00	1324.50

Total expenses for this report: 4196.92

Delete Line | Delete Report
Lookup . . .
Previous | Next
New Report | Save Report
Help | Exit

SUMMARY
Total Company Credit Card Charges 3473.23
Total Cash Expenses 723.69
Cash Advance (if any) 250.00
Due to Employee $473.69

Most traditional development environments carry a burden of assumed knowledge that keeps most non-technical users at arm's length. When the topic shifts to object-oriented development, even veteran programmers can turn pale. ObjectVision 2.0 provides an object-oriented development environment that assumes only that you have held a pencil and filled out a pre-printed paper form at some point in your life. After passing that hurdle, you're ready for object-oriented application design that's simple enough for neophytes, yet robust enough to satisfy most professional application developers. ObjectVision 2.0's strength shows best when existing paper-based systems are being converted to a graphical PC-based environment.

The shift of tasks from paper to computer-based systems often occurs simply because the manual processes are too slow or can be improved by putting them on-line. Probably the ultimate example of such a task is the bane of every business traveler, the Employee Expense Report.

DEVELOPING THE APPLICATION

While no program can make filling out an expense report fun, ObjectVision 2.0 can make the process a lot less tedious, and makes development of just such an application far easier than any other development tool. EXPENSES.OVD, included with the ObjectVision 2.0 sample data files, provides an example of how quickly such an application can be completed in an object-oriented environment.

ObjectVision 2.0 provides three basic tools for creating applications: The Form tool, for creating and editing the user interface for your application; the Stack tool, for organizing the application; and the Link tool, for building and managing connections to database files. Development always begins with the Form tool, where the basic visual objects of applications are created and assigned their properties. Even the forms themselves are objects, with capabilities for responding to and creating actions within themselves or other objects anywhere within the application. Each form is assigned a name—for example, the first form in this application is called "Expense Report."

With the Form tool open, an icon bar at the top of the screen allows creation and insertion of the various objects that may be placed in a form—fields, buttons, tables, text, filled boxes, filled boxes with rounded corners, lines, and pictures. Objects are selected with a click of the left mouse button on the appropriate icon, and the default attributes of each type can be assigned with a click of the right mouse button. Similarly, once an object is placed in a form, its attributes can be changed by clicking on it with the right mouse button.

In practice, Field objects act much like fields in a database, and can be used as such when connected to a file using the Link tool. For data entry, each field can be assigned any one of 14 different types. Any attribute of a Field object, including its name, may be changed at anytime—and in true object-oriented fashion, name changes of Field objects are automatically reflected wherever they are referenced throughout an application.

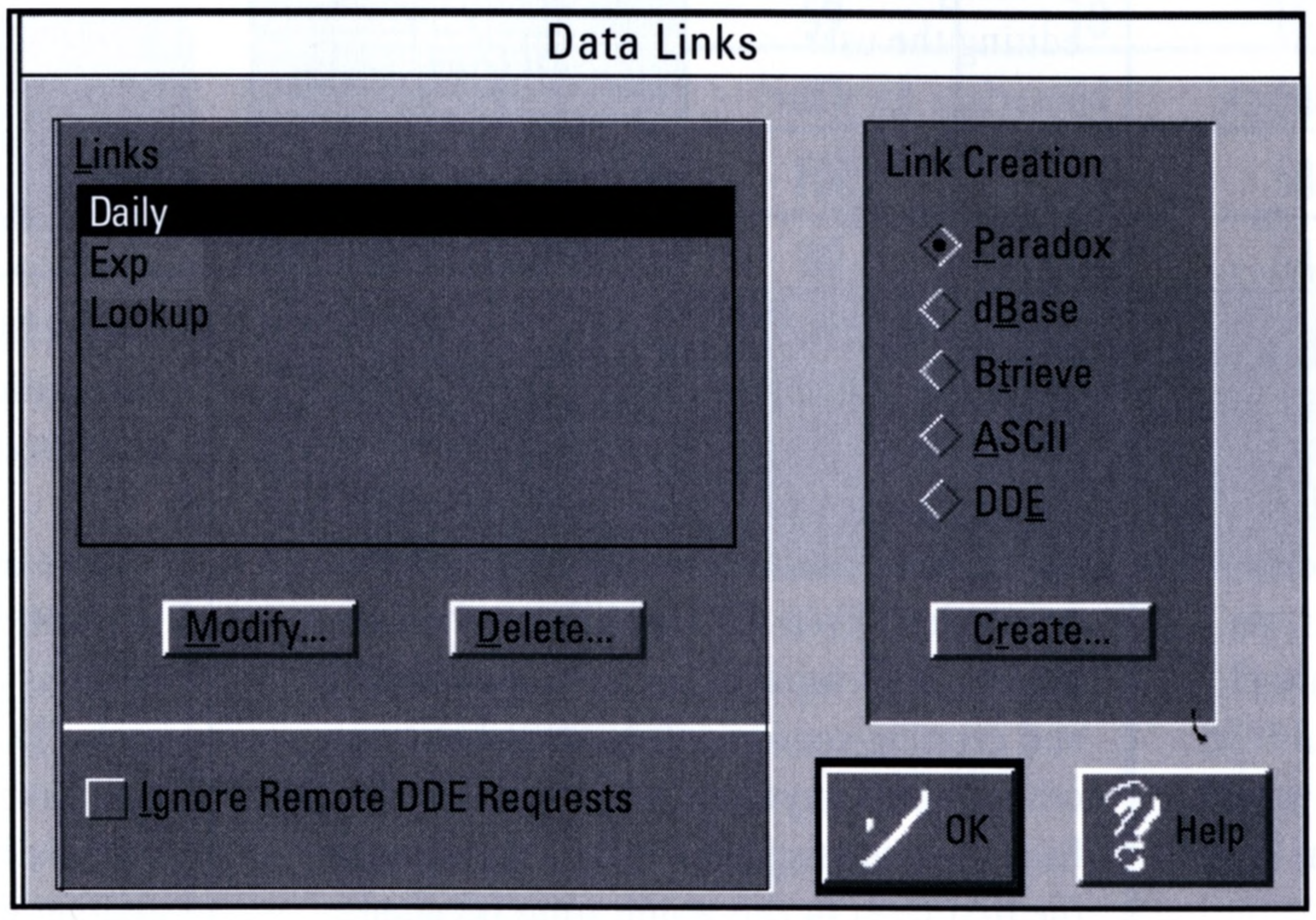
Table objects, new in ObjectVision 2.0, perform much as their name implies, grouping fields as columns, with multiple values within each. Just like any other field, each column may have its own unique properties—essentially, the columns are objects within another object. Tables can have any number of columns and rows and, when linked to a database, they can scroll vertically if all the records don't fit in the number of rows you define. Readers familiar with relational database applications will immediately recognize the usefulness of the combination of "regular" Field objects and Table objects. In a one-to-many relationship, the Field objects manage the data from the master record (the "one"), while the multiple-related records from the other file (the "many") are displayed through the Table object.

THE LINK TOOL

Using the Link tool, ObjectVision's Field and Table objects can connect to a number of database formats, including Paradox®, dBASE and others. If you have existing database files, that's great; if not, the Link tool can create them for you. Since there are several Date fields integral to the Expense Report application, Paradox links were used to take advantage of the Paradox Date field type.

When required, you can even perform database alchemy in ObjectVision 2.0, seamlessly relating files from differing formats.

When creating one-to-many links in ObjectVision 2.0, the order of link creation is vitally important—the main link must be created first in order for the relationship to function properly. The main link in Expense Report, called "Exp," links to the Paradox table EXPENSES, and stores general information about each expense report. To uniquely identify each report, the EXPENSES table is indexed on the "Employee" and "PeriodBeginDate" fields. This will allow multiple reports for each employee, providing that each has a unique reporting period.



The only other Field objects linked and stored in EXPENSES are Cost Center, Credit Card Expenses, Cash Expenses, and Cash Advance, all of which are connected in the Link Creation dialog box as both "OV Read" and OV Write" so that ObjectVision 2.0 can both read their values from the file and write new values into the database. The other Field objects are always calculated on the fly, so there's no need to store them in the database.

To complete the links, all the fields in the Table object (except Totals, which is automatically calculated) are linked with the DAILY Paradox table with the "Daily" link. As with the Exp link, all the table fields are defined as both OV Read and OV Write.

Additionally, in order to associate the records in Daily with the primary link, both the Employee Name and PeriodBeginDate fields are included in DAILY. Since the values of these two fields are already looked up and fed to the appropriate Field object by the Exp link, OV Read for both of them is not connected.

Each Link has a variety of options which are set by clicking on the "Options..." button in the Link Creation dialog box. The Exp link uses Auto Insert and Auto Update to automatically insert new records and store changes, and also employs Cascade Updates and Cascade Deletes to keep the records from the Daily link synchronized. The Locates for Exp are set so that Auto Locate is on, so that if values of an existing report are entered in the Employee Name and Report Date fields, that report is automatically displayed.

In the Daily link, only the Auto Insert and Auto Update options are activated. Like Exp, its Locates are set up to Auto Locate on Employee Name and PeriodBeginDate, and to use the "Restricted Range" option, which limits the records retrieved by Daily to only those which match the current lookup fields in the Exp link.

Value Trees

The easiest method of placing a calculated value in a Field object is to have the value computed by a Value Tree. Conceptually similar to programming flowcharts, Value Trees can have multiple decision nodes in order to reach their conclusion. Within its Table object, Expense Report employs Value Trees in the Auto Expenses and Totals fields. Since Value Trees in Table objects apply to an entire column, it's not necessary to enter the Value Tree in each cell in the column.

To allow a mileage expense of 24.5 cents per mile, Auto Expenses uses the simple value tree @ROUND(Auto Mileage*0.245,2). To add up the values in each row, the Value Tree for the Totals column reads +Auto Expenses+Airfare+ Hotel+Meals+Other. Directly below the Totals column is another Field object, Total Expenses. Its Value Tree employs the expression @COLUMNSUM(Totals), which adds up the values currently displayed in that column. Since Table objects can also be configured to allow scrolling of linked data, ObjectVision 2.0 provides the @LINKSUM function to add up all active values in one field of the link, not just those currently displayed in the Table object.

The two other Value Trees in Expense Report are in the SUMMARY box at the bottom of the form, and both reside in Field objects whose values will only appear as values are entered in the Total Cash Expenses and Cash Advance fields. Based on the values entered, the Amount Due is calculated and displayed as a non-negative number with the expression @ABS(Cash Advance-Total Cash Expenses). Directly to the left of Amount Due is the Field object Due to Whom. This Field object uses an unconditional branch in its Value Tree to compare the values of Cash Advance and Total Cash Expenses, determine if there is cash due to the company, to the employee, or neither, and create the appropriate label for Amount Due as its result.

Event Trees

Event Trees, unlike Value Trees, are not limited to Field and Table Objects, and are not constrained to passing a value to the object they are associated with. Perhaps the clearest example of triggering an event is within a Button object. All of the buttons in Expense Report capture the "Click" event, which means that when one of them is clicked, its Event Tree is processed. In addition to Buttons, the Click event can also be captured by Text and Graphic objects—useful for creating your own custom buttons with icons and "Click here..." messages.

Including an ampersand (&) character within a Button object's name causes the character following the ampersand to be underlined, giving the indication of a Windows-standard quick-key for the button. For example, the real name of the "Save Report" button is "&Save Report." To capture the Control-S and cause it to activate the button, the "Ctrl+S" event is captured in the application stack (right-click on the uppermost "ObjectVision" title bar to access). The sole expression in its conclusion node is @EVENT("&Save Report","Click"), which sends the Click event to the &Save Report button, which in turn evaluates its Event Tree.

Aesthetic Appeal

ObjectVision 2.0 provides an array of objects that can be added for visual appeal. Expense Report includes two sets of overlapping rounded rectangles, with the bottom rectangles filled with a darker color and

Borland doesn't give you just one or two ways to program faster.

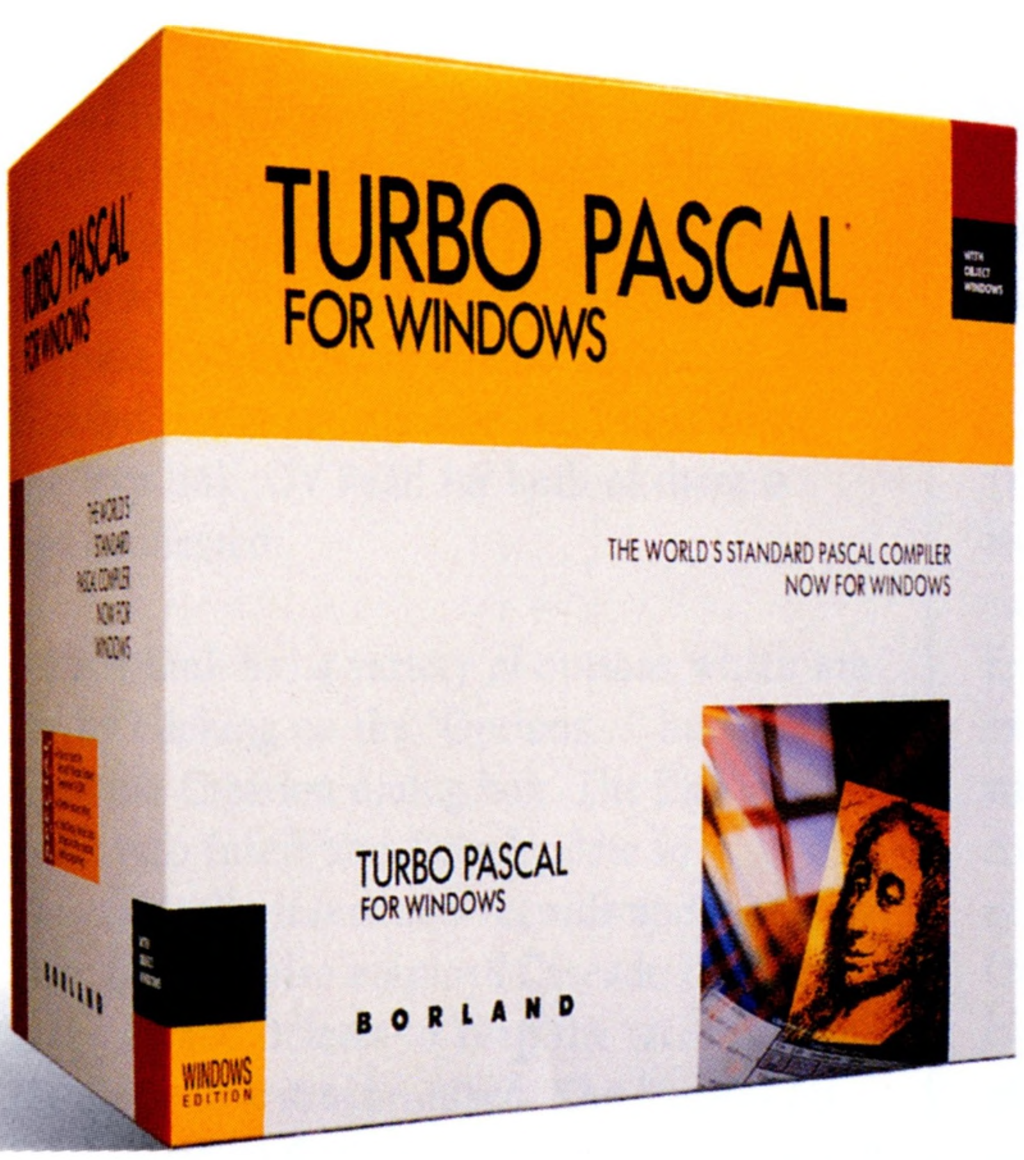
$149.95
UPGRADE FROM TURBO PASCAL: $69.95

$99.95
UPGRADE FROM TURBO C: $59.95

$299.95
UPGRADE FROM TURBO PASCAL: $99.95

We help you accelerate applications development every way possible.

Whether you're a novice or a pro, count on Borland to deliver precisely the tools you need to create better applications faster and easier.



WINTER SPECIAL
$99.95

Pascal programmers, please fasten your seat belts.

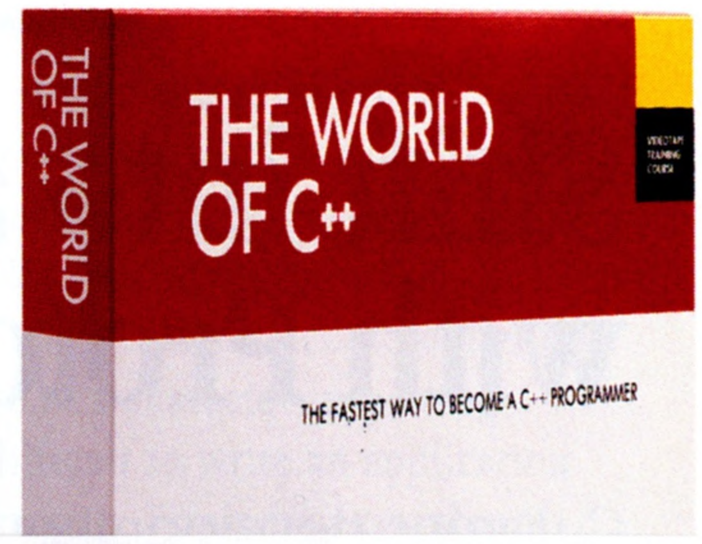
With Turbo Pascal 6.0, Pascal programmers can develop applications faster and easier than ever before.

It has Borland's Integrated Development Environment to streamline development, application frameworks for both DOS and Windows, and the incredibly fast Turbo Pascal compiler. In just a few lines of code, Turbo Vision gives you a complete DOS user interface, all in an event-driven architecture!

According to J.D. Power and Associates: *"Borland products rank as the "Best Application Software in Customer Satisfaction in Small, Medium-Sized and Large Businesses."* based on software capability, ease of use and customer support.

$39[95]

SPECIAL RATE FOR OBJECTVISION 1.0 OWNERS: $19[95]



Enter the

BORLAND LANGUAGE EXPRESS

SPRING-ALONG-THE-SEINE SWEEPSTAKES!

Borland makes your life easier with its wide array of powerful, easy-to-use software. And if you're lucky we'll help you enjoy a relaxing vacation in Paris. There are lots of other great prizes, too, so be sure to enter today!

FIRST PRIZE!

A WEEK FOR TWO IN PARIS!

Imagine yourself strolling along the Seine, shopping your way down the Champs Elysee, or just lingering over an espresso at a sidewalk cafe. Win the Sweepstakes and all this is yours.

As the winner, you and a companion will fly to Paris and stay for a week. Bon voyage!

3 WINNERS!

SECOND PRIZE

5 BORLAND PRODUCTS OF YOUR CHOICE

Choose any of five valuable products from the Borland product library. That includes exciting, new products like new *Borland C++ & Application Frameworks 3.0*, new *ObjectVision 2.0*, new *Turbo C++ for Windows, Turbo Pascal for Windows, Paradox Engine*...you get the idea. It's nearly a $3,700 value!

30 WINNERS!

THIRD PRIZE

YOUR FAVORITE BORLAND LANGUAGE PRODUCT

Choose any Borland language product, valued up to $749!

2 EASY WAYS TO ENTER

You can use the order form attached or the postage-paid reply card on the outer cover. Whichever you use, be sure to fill it out completely. When you check the box showing your area of interest and mail it back to Borland, you'll be automatically entered to win!

OFFICIAL RULES
No Purchase Necessary

1. How to Enter: To enter the Borland Language Express "Spring-Along-The-Seine Sweepstakes," completely fill out and return either the order form or postage-paid reply card. At least one box must be checked to indicate your area of interest. All entries must be postmarked by April 1, 1992 and received by April 10, 1992. Entries that are mechanically reproduced, forged, mutilated, defaced or altered in any way are not eligible. All entries become the property of Borland International, Inc.

2. Judging: Winners will be selected at random on April 16, 1992 from all entries received at sweepstakes headquarters. Neither sponsor nor Marden-Kane, Inc. is responsible for late, lost or misdirected entries or postage-due mail. Only one prize per person, family or household will be awarded. All prizes will be awarded. Odds of winning depend upon the number of entries received.

This sweepstakes is under the supervision of Borland International, Inc., an independent judging organization who reserves the exclusive right to interpret all conditions in regard to this promotion without claim for damage or recourse of any kind. By participating in the sweepstakes, entrants agree to be bound by the rules and the decisions of the judges which shall be final.

3. Notification: Winners will be notified by mail on or about April 30, 1992 and will be required to sign an Affidavit of Eligibility and Publicity/Liability Release which must be returned within 14 days from date of notification. If the affidavit is not returned within this time period properly executed, or is returned from the post office as undeliverable, an alternate winner(s) will be selected. Winners grant permission to the use of their name, photography/likeness for advertising and promotion for this and similar promotions without additional compensation.

4. Prizes: (1) First Prize: One week in Paris, including airfare and hotel accommodations. Coach airfare for the winner and a companion to Paris. Departure is from the nearest major airport of Borland's choice. Trip is to be made during a specified time. Accommodations at a hotel of Borland's choice for 7 days and 6 nights. No substitutions. (3) Second Prizes: 5 different products selected from the entire Borland library of software products. ARV: $3,700 (30) Third Prizes: 1 Borland language product. ARV: $749.

5. General Conditions: All taxes are the responsibility of the winner. There are no prize substitutions permitted, nor are prizes transferable.

Winners accepting prizes agree that all prizes are awarded on the condition that Borland International, Inc. and their agents, representatives and employees will have no liability whatsoever for any injuries, losses, or damages of any kind resulting from acceptance, possession or use of the prize. All prizes must be claimed within 6 months of being notified as being a winner, or the prize will be forfeited.

6. Eligibility: Sweepstakes open to all persons 18 years or older who are *residents of the United States* except employees and their immediate families of Borland International, Inc., its subsidiaries, affiliates, advertising and promotion agencies and Marden-Kane, Inc. Void where prohibited by law.

7. Winners List: For the names of the winners, available after May 7, 1992, send a self-addressed, stamped envelope to Borland International Sweepstakes Winners, Borland International, Inc. Attn: Brian Anderson, 1800 Green Hills Road, Scotts Valley, CA 95067.

B O R L A N D

BORLAND TECHNICAL DEVELOPMENT CENTER

1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0005

❑ *Yes!* Please enter my name in the BORLAND LANGUAGE EXPRESS SWEEPSTAKES.

❑ *Yes!* Please send me information on the *'92 Borland International Developers Conference.*

REGISTERED TO:

ZAE03
DAVID WILLIAMS
14222 KIMBERLY 421
HOUSTON TX 77079

❑ *If this is not your correct name or address, please check here and make necessary corrections above or on the back of this order form.*

TO ORDER, MAIL THIS FORM OR CALL

1-800-331-0877
OR FAX
1-408-439-9119
(24 HRS)
or see your software dealer

* Make checks payable to Borland International, Inc. U.S. funds only. We do not accept CODs. Not good with any other offer from Borland. Offer good in U.S. only.
** Freight fee in U.S. is $15 for up to 3 products, add $5 for each additional product.
*** Sales tax applicable in the following states: CA, CO, CT, DC, FL, GA, IL, MA, MD, MI, MN, MO, NC, NJ, NY, OH, PA, TN, TX, UT, VA, WA. For residents of CO, MI, NY, PA and TX, sales tax on freight charges must be included. Purchase orders accepted upon approval — $150 minimum. Terms net 30 days. Attach this order form to a preprinted P.O. form. Sweepstakes winners will be notified by telephone and must comply to official Sweepstakes rules. For a copy of official Sweepstakes rules, contact Borland Language Express, 1800 Green Hills Road, Scotts Valley, CA 95066. Void where prohibited by law.

		UPGRADE/SPECIAL	NEW PURCHASE	TOTAL
BORLAND® C++ 3.0 Product or serial number for upgrade:	Upgrade from: ❑ BC++ ❑ TC Pro ❑ TC++ Pro	$125.00	$495.00	$
BORLAND® C++ & APPLICATION FRAMEWORKS 3.0 Product or serial number for upgrade:	Upgrade from: ❑ BC++ ❑ TC Pro ❑ TC++ Pro	$199.95	$749.00	$
	Upgrade from BC++ & AF 2.0	FREE!		
TURBO C®++ FOR WINDOWS 3.0 Product or serial number for upgrade:	Upgrade from ❑ TC++ ❑ TC	$ 89.95	$149.95	$
	Upgrade from ❑ TPascal ❑ dBASE	$ 99.95		
TURBO C®++ 2nd Edition Product or serial number for upgrade:	Upgrade from TC	$ 59.95	$ 99.95	$
TURBO C®++ & TURBO VISION Product or serial number for upgrade:	Upgrade from TC++	$ 99.95	$199.95	$
TURBO PASCAL® 6.0 Product or serial number for upgrade:	Upgrade from any Turbo Pascal product	$ 69.95	$149.95	$
TURBO PASCAL PROFESSIONAL® 6.0 Product or serial number for upgrade:	Upgrade from any Turbo Pascal product	$ 99.95	$299.95	$
TURBO PASCAL® FOR WINDOWS		N/A	$ 99.95 WINTER SPECIAL!	$
OBJECTVISION™ 2.0 Product or serial number for upgrade:	Upgrade from ObjectVision 1.0	$ 49.95	$ 99.95 WINTER SPECIAL!	$
VISIONS® BORLAND VIDEO TRAINING SERIES "THE WORLD OF C++"		N/A	$ 99.95	$
"WORLD OF OBJECTVISION" ObjectVision 1.0 serial number for upgrade:	Special price for ObjectVision 1.0 owners only	$ 19.95	$ 39.95	$
PROTOGEN from PROTOVIEW for all Turbo C++ and Borland C++ owners.		N/A	$ 49.95 WINTER SPECIAL!	$

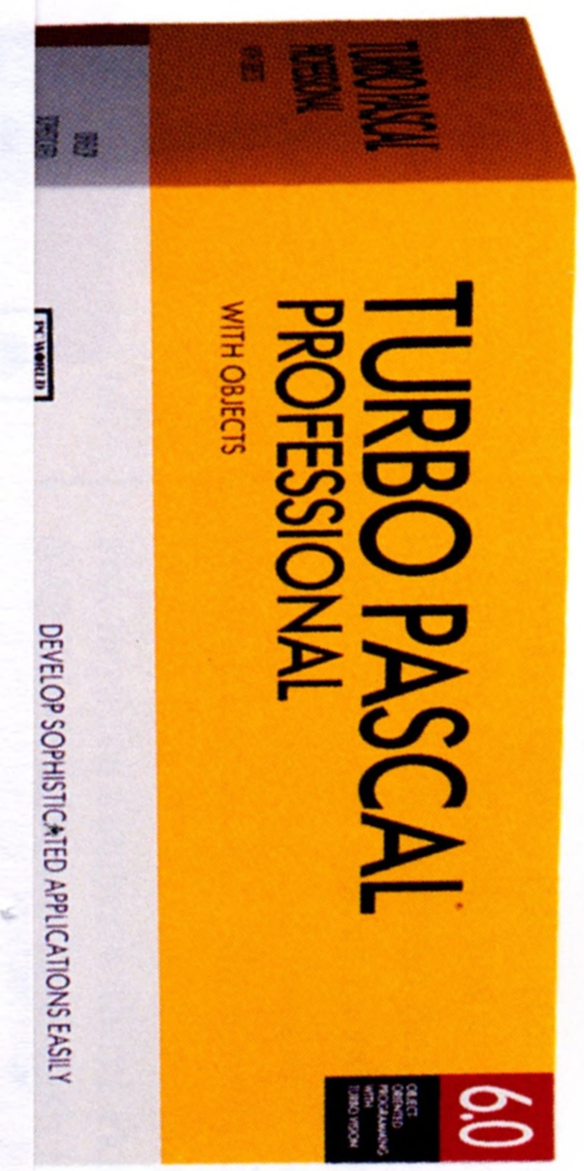
ORDERING INSTRUCTIONS

1. Specify disk size: ❑ 5.25" ❑ 3.5"
2. Indicate payment type:
   ❑ Check or Money Order (U.S. funds only)*
   ❑ VISA ❑ MasterCard ❑ American Express

Account No: ______
Expiration: ___/___
Signature: ______

SUBTOTAL	$
****FREIGHT**	$
*****SALES TAX**	$
TOTAL DUE	$

PLEASE DETACH AND ENCLOSE IN REPLY ENVELOPE PROVIDED.

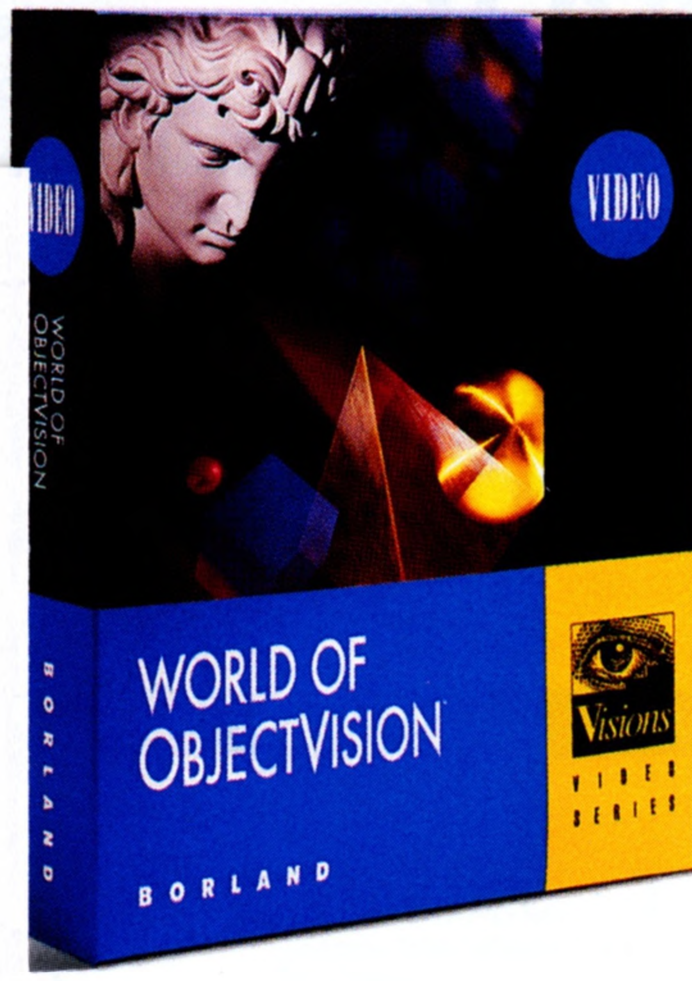


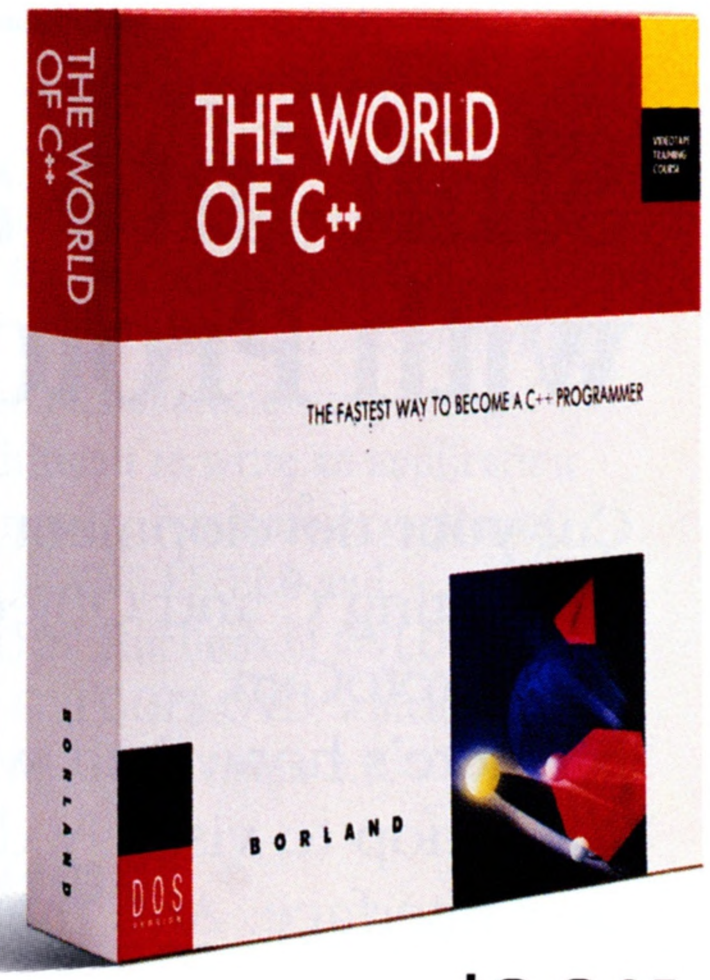
$299.95
UPGRADE FROM TURBO PASCAL: $99.95

HAVE YOU INCLUDED YOUR CHECK OR MONEY ORDER?
PLEASE BE SURE TO MOISTEN ENVELOPE FLAP.

$39⁹⁵

SPECIAL RATE FOR OBJECTVISION 1.0 OWNERS: $19⁹⁵





$99⁹⁵

'urn your TV into a teacher: he *Visions* video training courses.

To help you supercharge your programming kills, Borland presents these superb vehicles: Vinner of a *Byte Magazine 1991 Award of Merit*, The World of C++" video helps you learn the vorld's most popular object-oriented language in e comfort of your living room. It comes omplete with two videos, workbook and xamples with source code that you can use in our programs.

And the "World of)bjectVision" video xplains this powerful ew way of program- ing without writing ode! Both are absolute must sees."

To place orders, call our upgrade line at 1-800-331-0877 or use the enclosed order form.
Remember to specify disk size when making phone orders. Offer good in the U.S. only. All Borland software comes with our no-questions-asked, 60-day money-back guarantee.

BORLAND

'O ORDER CALL TODAY 1-800-331-0877

B O R L A N D

BORLAND TECHNICAL DEVELOPMENT CENTER
1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0005

❑ *Yes!* Please enter my name in the BORLAND LANGUAGE EXPRESS SWEEPSTAKES.
❑ *Yes!* Please send me information on the *'92 Borland International Developers Conference.*

REGISTERED TO:

ZAED3
DAVID WILLIAMS
14222 KIMBERLY 421
HOUSTON TX 77079

❑ *If this is not your correct name or address, please check here and make necessary corrections above or on the back of this order form.*

TO ORDER, MAIL THIS FORM OR CALL

1-800-331-0877
OR FAX
1-408-439-9119
(24 HRS)
or see your software dealer

* Make checks payable to Borland International, Inc. U.S. funds only. We do not accept CODs. Not good with any other offer from Borland. Offer good in U.S. only.
** Freight fee in U.S. is $15 for up to 3 products, add $5 for each additional product.
*** Sales tax applicable in the following states: CA, CO, CT, DC, FL, GA, IL, MA, MD, MI, MN, MO, NC, NJ, NY, OH, PA, TN, TX, UT, VA, WA. For residents of CO, MI, NY, PA and TX, sales tax on freight charges must be included. Purchase orders accepted upon approval — $150 minimum. Terms net 30 days. Attach this order form to a preprinted P.O. form.
Sweepstakes winners will be notified by telephone and must comply to official Sweepstakes rules. For a copy of official Sweepstakes rules, contact Borland Language Express, 1800 Green Hills Road, Scotts Valley, CA 95066. Void where prohibited by law.

		UPGRADE/SPECIAL	NEW PURCHASE	TOTAL
BORLAND® C++ 3.0 Product or serial number for upgrade:	Upgrade from: ❑ BC++ ❑ TC Pro ❑ TC++ Pro	$125.00	$495.00	$ ________
BORLAND® C++ & APPLICATION FRAMEWORKS 3.0	Upgrade from: ❑ BC++ ❑ TC Pro ❑ TC++ Pro	$199.95		

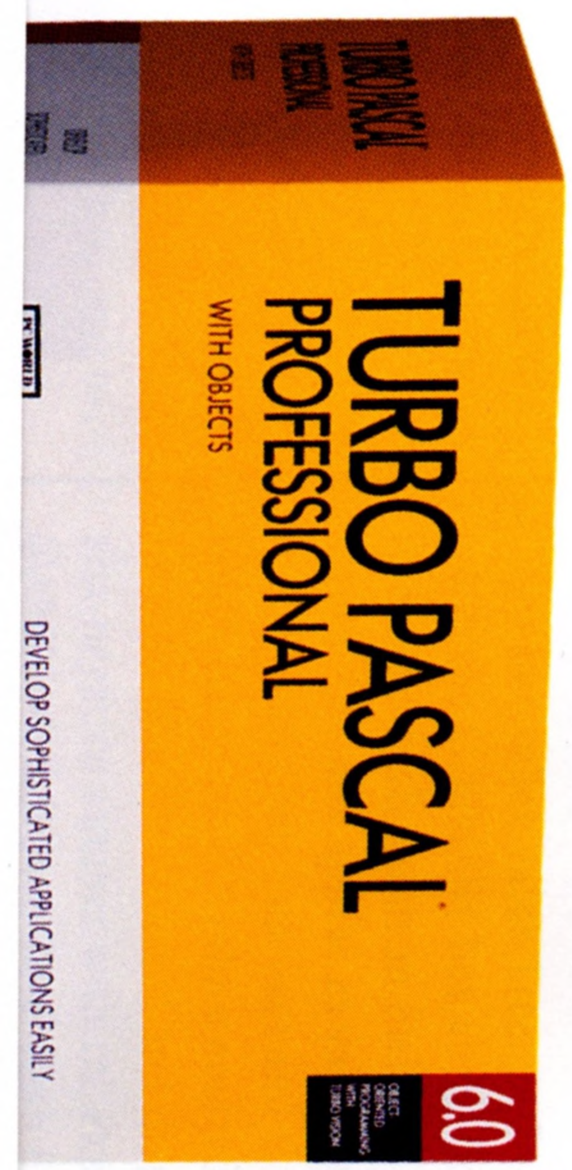
ATTN: *Order Processing*
PLEASE EXPEDITE
BLX4

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 200 SANTA CRUZ, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

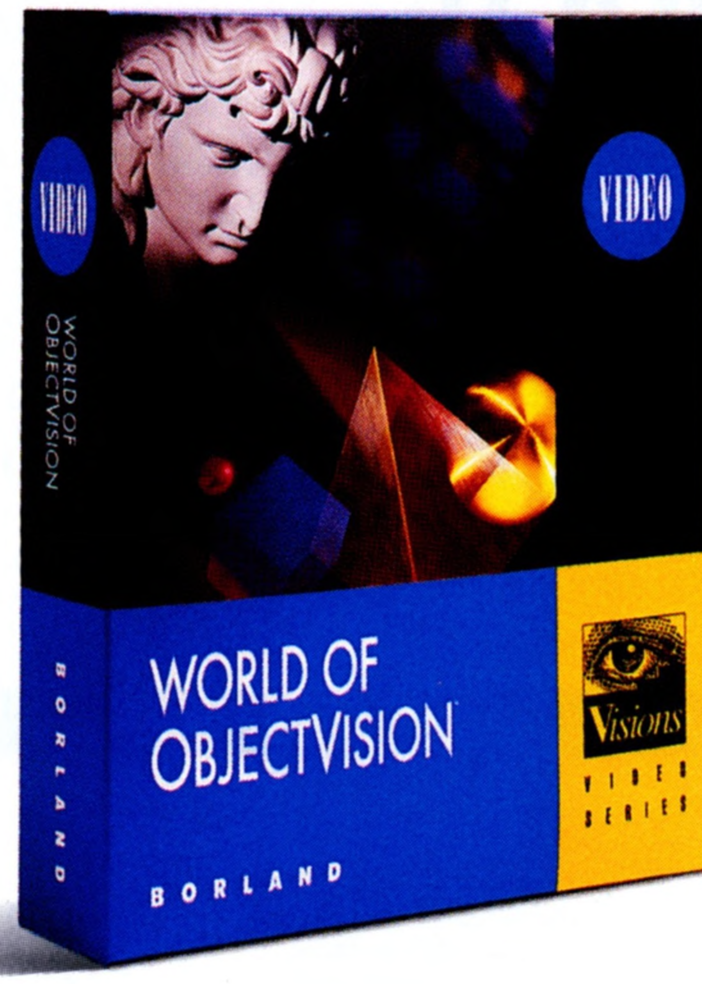
B O R L A N D
TECHNICAL DEVELOPMENT CENTER
1800 Green Hills Road
P O Box 660005
Scotts Valley CA 95067-9985

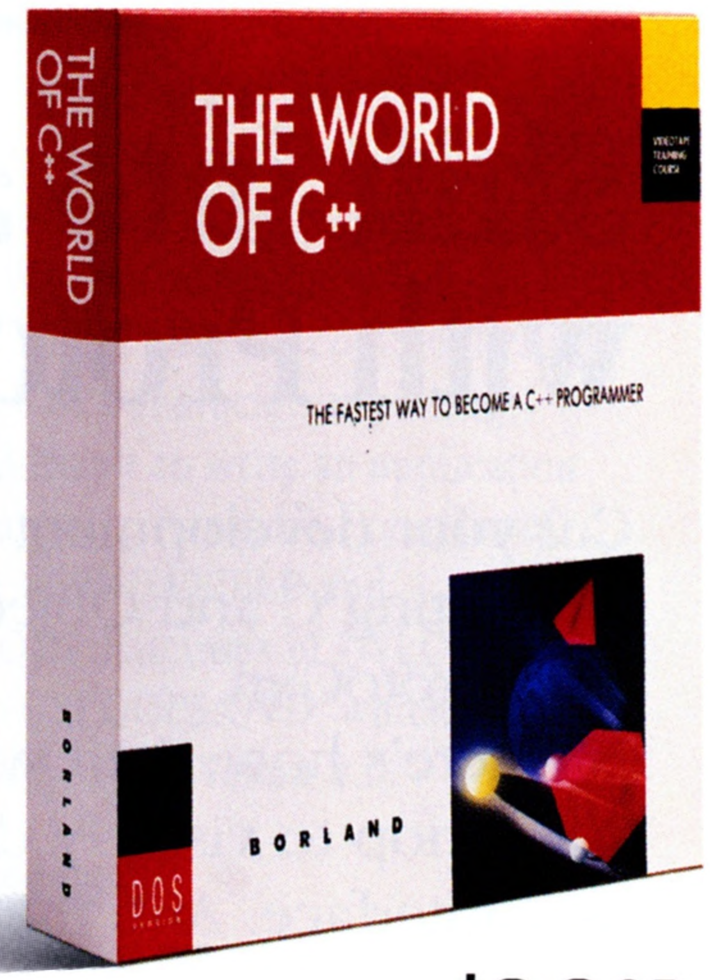


$299.95
UPGRADE FROM TURBO PASCAL: $99.95

$39.95

SPECIAL RATE FOR OBJECTVISION 1.0 OWNERS: $19.95





$99.95

To push application development to the max, choose Turbo Pascal Professional 6.0 — with its high-powered Turbo Debugger, Turbo Profiler and Turbo Assembler.

And if you're migrating to Windows, get Turbo Pascal for Windows. It comes complete with the world's fastest Pascal compiler, our new Object-Windows application framework, and more.

How to get an "A+" in C and C++.

Turbo C++, the fastest and easiest way to learn C or C++, has everything you need to succeed. Tutorials for both C and C++, the Programmer's Platform that integrates all the tools to edit, compile, debug and run your applications. An intuitive hypertext help system. Exhaustive documentation and much, much more.

But Turbo C++ isn't just for beginners. It's a high performance tool with a full-featured compiler and advanced capabilities you can grow into.

Even better, Turbo C++ is available with Turbo Vision. This application framework gives you sophisticated, ready-made user interfaces with just a few lines of code. It's an incredible time saver.

Whether you're just learning to program in C or graduating to object-oriented C++ — Turbo C++ is for you.

$199.95

UPGRADE FROM TURBO C++: $99.95

Turn your TV into a teacher: the *Visions* video training courses.

To help you supercharge your programming skills, Borland presents these superb vehicles: Winner of a *Byte Magazine 1991 Award of Merit*, "The World of C++" video helps you learn the world's most popular object-oriented language in the comfort of your living room. It comes complete with two videos, workbook and examples with source code that you can use in your programs.

And the "World of ObjectVision" video explains this powerful new way of programming without writing code! Both are absolute "must sees."

To place orders, call our upgrade line at 1-800-331-0877 or use the enclosed order form.

Remember to specify disk size when making phone orders. Offer good in the U.S. only. All Borland software comes with our no-questions-asked, 60-day money-back guarantee.

BORLAND

TO ORDER CALL TODAY 1-800-331-0877

Shift into automatic with ProtoGen.

Cut your development time by automatically generating C and ObjectWindows code with ProtoGen.

Here's how: Start with Borland's Resource Workshop to visually design the elements of your user interface. And then shift into ProtoGen and automatically create the code to link those elements!

After ProtoGen produces the framework for your application, you simply add the application specific code and recompile. Creating user interfaces was never so easy!

SPECIAL

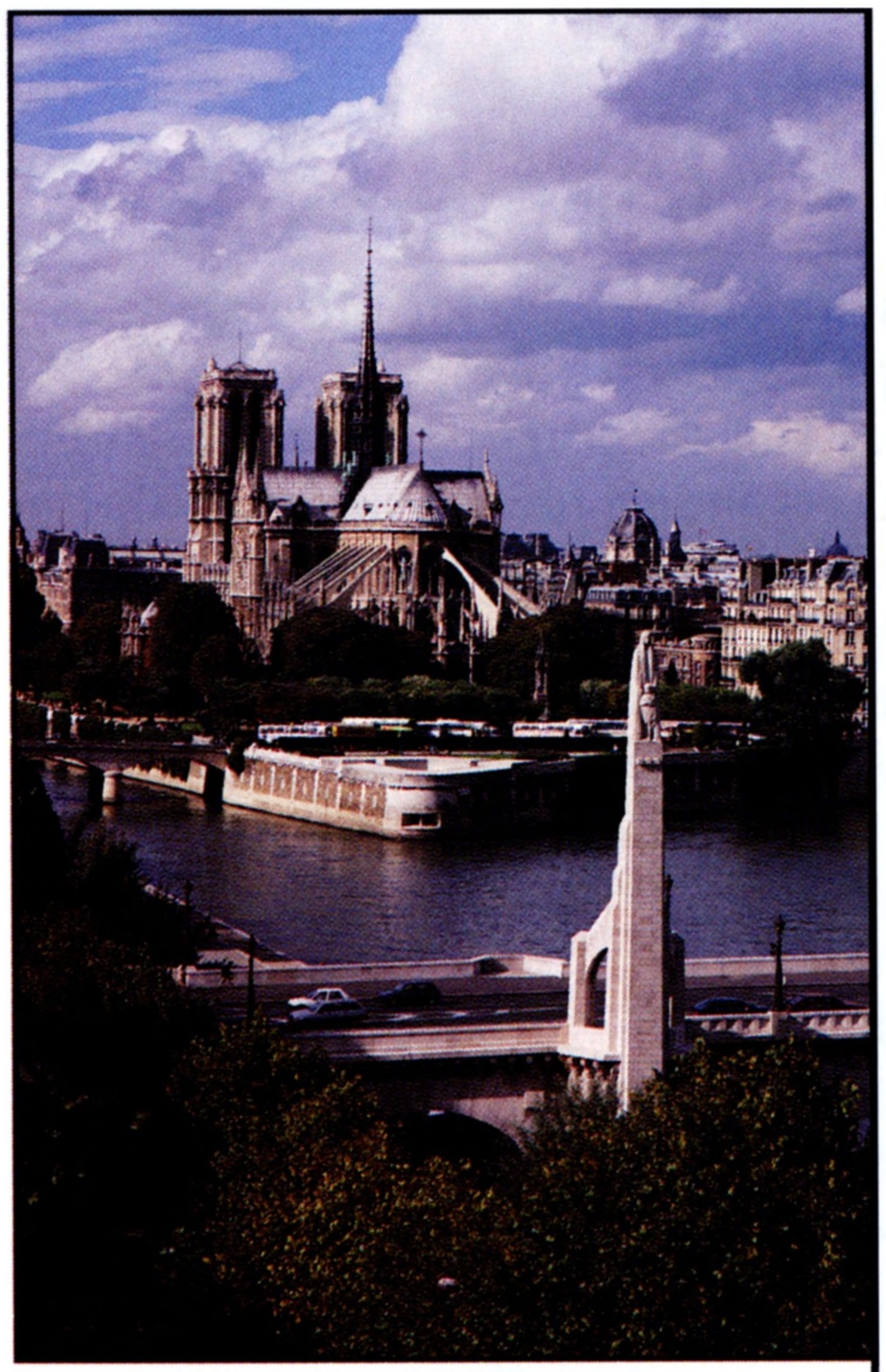
Right now, as a Borland customer, you can get this powerful, new $495.95 tool at 1/10th its regular price — just $49.95!

It's an offer that's too good to refuse.

Enter the

BORLAND *LANGUAGE* EXPRESS

SPRING-ALONG-THE-SEINE SWEEPSTAKES!



Imagine yourself... strolling along the Seine, shopping your way down the Champs Elysee or just lingering over an espresso at some sophisticated sidewalk cafe.

FIRST PRIZE

A week for two in Paris.

3 WINNERS!

SECOND PRIZE

Choose any 5 products from the family of Borland software products.

30 WINNERS!

THIRD PRIZE

Your choice of any Borland language product.

WIN A WEEK FOR TWO IN PARIS OR VALUABLE BORLAND SOFTWARE.

Programming shouldn't be all work and no play. Continue receiving your FREE issue of *Borland Language Express* and you could win an exciting week for two in Paris! Just return the Order Form inside this issue–or the Return Card on the outside cover. And when you check the box that shows your area of interest, you're automatically entered to win!

slightly offset to the right and bottom to create a shaded effect. Since ObjectVision 2.0 offers a full 16-color palette, plus shading options, creating effective color combinations is quite simple.

The comical character in the upper left corner is copied directly from a clip-art library and pasted into ObjectVision 2.0 as a graphic object, stored in an .OVG file. The primary advantage of storing graphic objects in .OVG files, with simple pointers to them in the .OVD, is that they can be reused in the various forms in a single application. Each .OVG can be sized similarly or differently and also may be used in several other applications without the file-size increase you would expect.

DLLs and ObjectVision 2.0

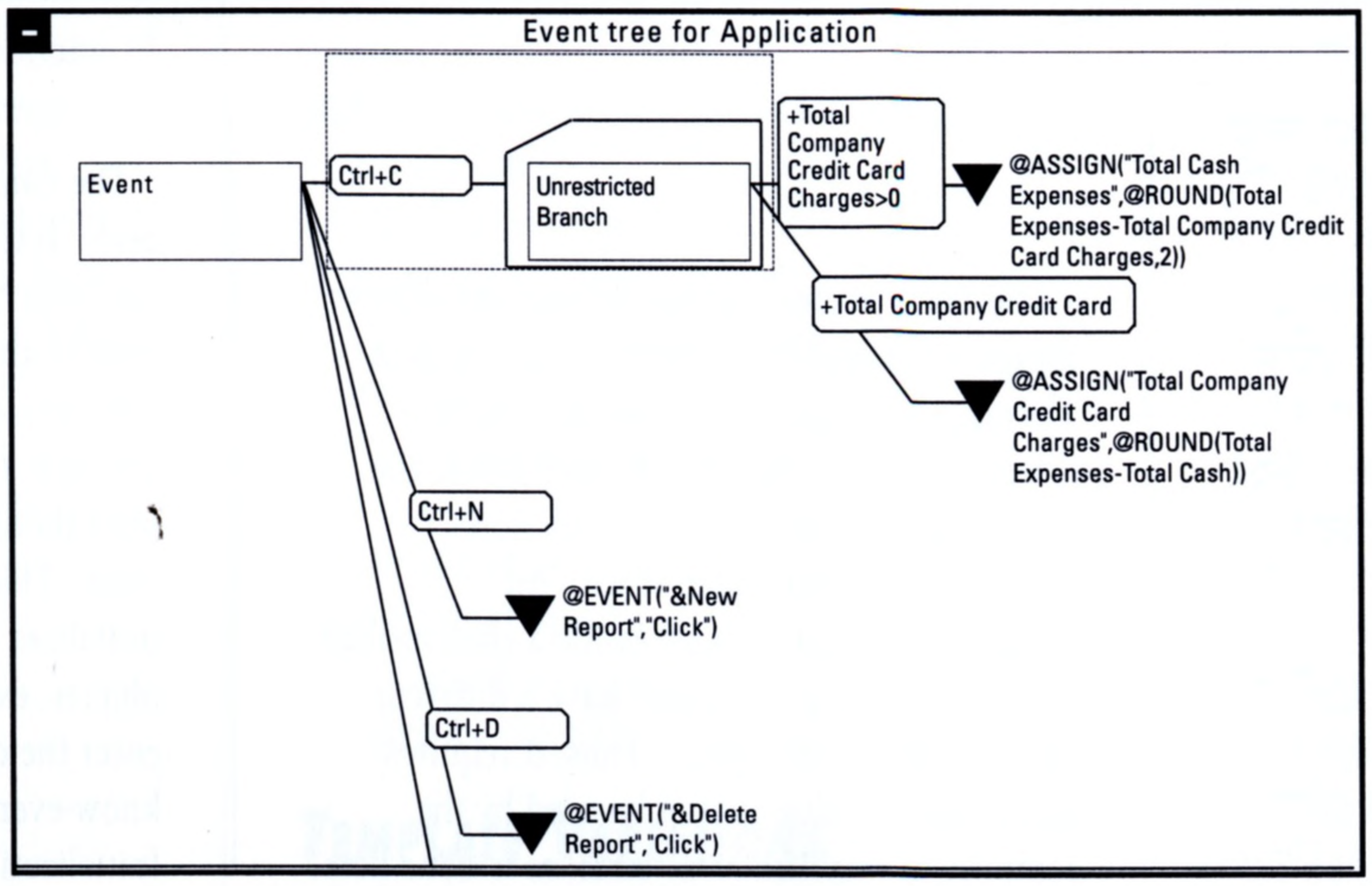
For a look at using DLLs in Object Vision 2.0, we turn to another of the sample applications included with ObjectVision 2.0—HANGMAN.OVD. Two DLLs are used by the Hangman game; one written in Turbo Pascal for Windows to provide a random number generator, the other written in Borland C++ to provide a WHILE looping function.

The source for RAND.PAS and OVLOOPS.C are shown in Listings 1 and 2, respectively. Since both programs are relatively short and self-documenting, I will leave you to explore the source code on your own, and will devote these last few paragraphs to show how and why they are useful in OV2.0.

To use a DLL in an OV2.0 application, you first must connect it by using the @REGISTER function. Both RAND and OVLOOPS are declared in the event tree of the application, automatically executing when the application is opened. As you can see in Figure (A), RAND is declared using the normal @REGISTER syntax, while OVLOOPS is declared slightly differently. Since OVLOOPS.DLL contains several functions, it was built with a special function @REGISTER_OVLOOPS, which is designed to automatically register the other functions.

One of the integral aspects to the Hangman game is that the puzzle must be looked up from a Paradox table in a random fashion, so that puzzles will rarely come up in any particular order. All the puzzles in HANGMAN.DB are indexed by a numeric field, so looking up any particular puzzle with @LOCATE and a random number generated by @RAND is a simple task. But there's a catch—@RAND returns a decimal number between 1 and 0, and to locate a puzzle we need an integer between 1 and the number of records in the "Phrase" link. To accomplish this, the following equation is used:

@LOCATE("Phrase", 0,
 @INT(@RAND
*@LINKCOUNT("Phrase")))



The @WHILEFIELD function in OVLOOPS.DLL is used by Hangman in several places. First, it is employed in the "Click" event in the "New Game" button as follows:

@ASSIGN(YesNo,Yes)
@WHILEFIELD("YesNo",
 "FillEvent")

As long as the value of the field "YesNo" remains "Yes," this function will continue to increment the value in the "FillEvent" field, activating its "Change" event. In this case, this event has a series of instructions that builds a string identical to the puzzle's answer, except that each alphabetic character is replaced by a @CHAR(127). You can view the Event Tree of FillEvent by selecting the "Scratch" form while in the Form tool.

Also in the Scratch form is a field object called "ReplaceEvent," which uses @WHILEFIELD to replace each @CHAR(127) with the appropriate alphabetic character when it is chosen—serving to reveal all instances of the chosen character. Without any type of WHILE function, a single function could only find one instance of a character in a string, making it difficult to write an application such as Hangman. For more examples of how to use @WHILEFIELD and @WHILELOOP functions of OVLOOPS.DLL, see the file OVLOOPS.OVD, which is included with ObjectVision 2.0.

Conclusion

For those who've never developed applications in an object-oriented environment, ObjectVision 2.0 is the perfect first step, since it gives access to existing data (in Paradox or dBASE) and lets your imagination and creativity take over from there. Those familiar with the original ObjectVision will find the improvements in version 2.0 appealing, as they add more color and a more robust development environment. And while it's not a hardcore programming tool all by itself, programmers will appreciate its easy DLL connectivity, the speed of application development, and the flexibility it puts into their users' hands.

About the Author

David Fail is Technical Product Manager for Application Methods Inc., based in Seattle, WA, where he developed several of the sample applications included with ObjectVision 2.0, including Expense Report and Hangman.

templates in Borland C++

BY BRUCE ECKEL

C++ programming really isn't so hard. You make some objects, and you send messages to them. But sometimes you need a place to stash objects while you're working with them. In C, an array is traditionally used for this, but arrays are primitive and don't provide much help or safety in problem-solving. And an array is a fixed size— what about situations where you don't know how many objects you'll have?

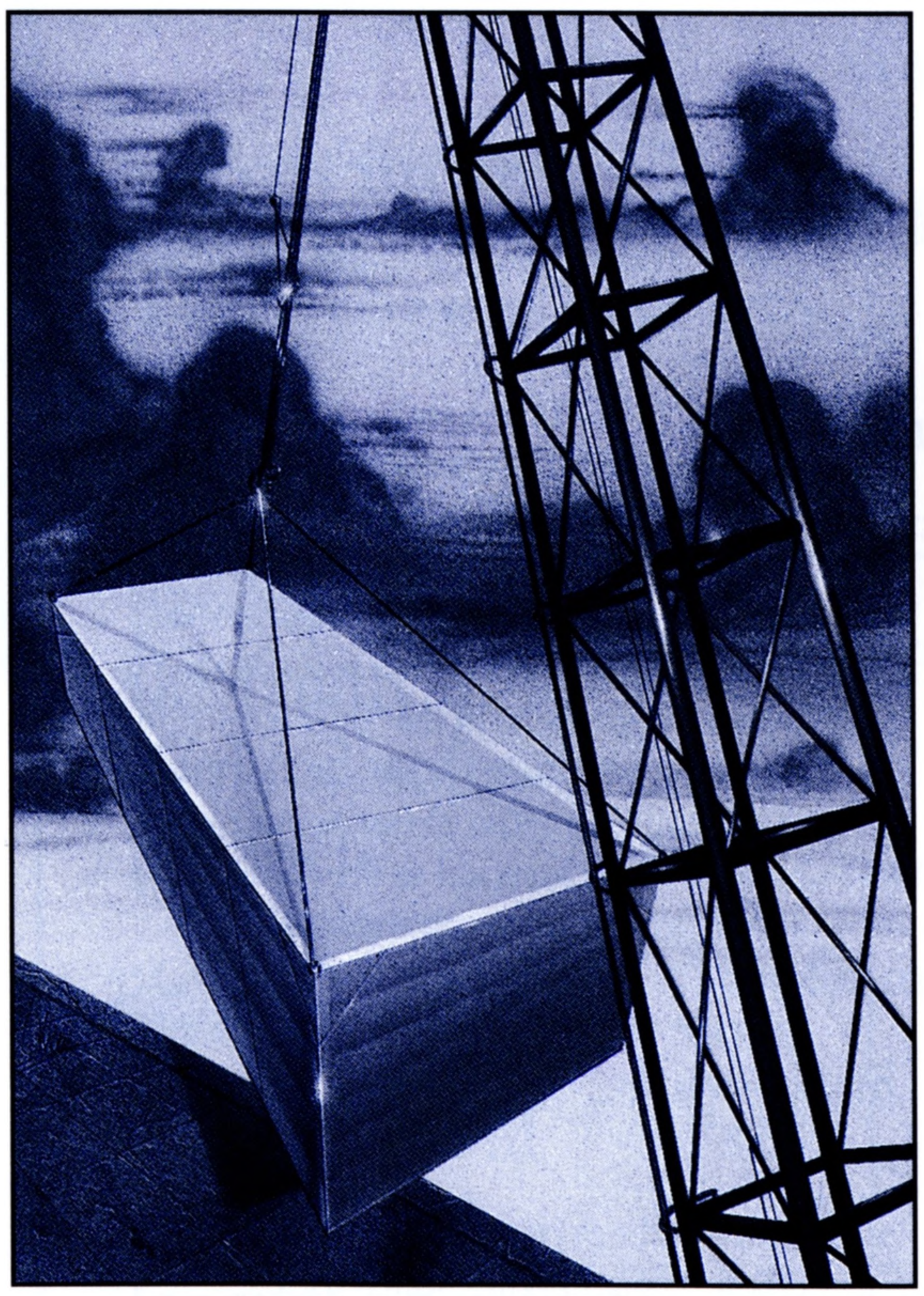
The C++ solution is a special type of class, alternately called a collection or a container class. This is a class which holds objects (or pointers to objects) of other classes. Version 3.0 of the C++ language incorporates the new template feature added to the draft ANSI C++ specification. In this article, we'll look at why you need templates, and how to use them for generic container classes and generic functions, as well as exploring something called an iterator.

Container Classes

Let's look at a very simple example of a container class, one which simply holds pointers to ints:

```
class queue {
  enum { size = 10 };
  int *q[size];
  int in, out; // indexes
public:
  queue() : in(0), out(0) {}
  void add(int * i) {
    q[in++] = i;
    if(in >= size) in = 0;  // wrap
  }
  int* get() {
    if(out == in) return 0;
    int *result = q[out++];
    if(out >= size) out = 0;
    return result;
  }
};
```

This class creates a queue, so you can retrieve things in the same sequence you put them in. The size of the queue is established by the enumeration size, which is hidden inside the class — a good thing to do so you don't "pollute" the global name space. An enumeration must be used inside a class instead of a const, since a const may have a different value for different objects. Thus, it requires space, and its value cannot be used by the compiler to establish array sizes, as is done here.



The constructor simply sets the two indexes to zero. It does this in the constructor initializer list, which is the place where all member objects and base-class objects (if you're using inheritance) are initialized. The initialization for these two ints looks like constructor calls, even though int is a built-in type and not a class. This is allowed in the constructor initializer list so you can initialize all member objects, even if they're built-in types, before you enter the constructor body. That way, you know everything has been set up properly. Initialization of built-in types this way simply means assignment.

Notice that the implementation of the queue is hidden, and it isn't really important. You could use a linked list if you wanted, and the user wouldn't see any difference in the interface. In addition, you would certainly want more error-checking code (which is left out here for the sake of brevity).

Here's a test for the queue. Note that it stores pointers, not the objects themselves:

```
int I[] = { 1, 3, 5, 7, 11, 13 };
queue iq;
for(int i = 0;
    i < sizeof(I)/sizeof(I[0]);
    i++)
  iq.add(&I[i]);
for(int* p = iq.get();
    p; p = iq.get())
  cout << "iq.get() = " << *p << endl;
```

In the definition for I, the compiler automatically counts the number of elements in the initialization list and allocates the proper amount of memory. When printing,

the for loop calls get() until a zero pointer is returned to indicate the queue is empty. Notice the iostreams library is being used for printing to standard output (cout).

Now this is all very nice, but chances are you probably have your own class you want to store in a container. Traditionally, you have two choices. You can take my queue and adapt it for your new class, and adapt it again for other new classes. Or you can automate the process using the preprocessor to make substitutions in a big macro. Debugging a macro is much more difficult than a template, since the compiler expands a macro into a single line where it is used, while it goes back and points you at the error in a template.

Fortunately, people realized how important container classes are, and how the language needs to support them directly rather than forcing programmers to flounder around with the preprocessor. Thus the template was born, and is now implemented in Borland C++ 3.0.

A Template Example

To see how they work, here's the queue implemented with templates:

```
template<class T> class queue {
  enum { size = 10 };
  T *q[size];
  int in, out; // indexes
public:
  queue() : in(0), out(0) {}
  void add(T * i) {
    q[in++] = i;
    if(in >= size) in = 0;  // wrap
  }
  T* get() {
    if(out == in) return 0;
    T *result = q[out++];
    if(out >= size) out = 0;
    return result;
  }
};
```

Notice that everywhere we stored an int, we now use a placeholder called T (although the name is arbitrary). At compile-time, the compiler will replace T with whatever type you specify. To accomplish this, you must precede the class with template<class T>, which means this isn't an ordinary class, and that the special placeholder will be called T.

That's all there is to it. Now if we want to create a new type of queue, it's trivial. Here's how to create the int queue we used before:

```
queue<int> iq;
```

VERSION 3.0 OF THE C++ LANGUAGE INCORPORATES THE NEW TEMPLATE FEATURE ADDED TO THE DRAFT ANSI C++ SPECIFICATION.

And we can also make a queue for Strings:

```
  queue<String> sq;
And test it like this:
  String S[] = { "this", "is", "a",
                 "queue", "too" };
  for(i = 0;
      i < sizeof(S)/sizeof(S[0]);
      i++)
    sq.add(&S[i]);
  for(String* sp = sq.get();
      sp; sp = sq.get())
    cout << "sq.get() = " << *sp << endl;
```

Template Variations

So far, the queues have been a fixed size, determined by the local enum size. But you may want to change the size, and it seems messy to edit the class to do this. Fortunately, you can have more than one argument in a template, and you aren't restricted to only using class arguments. For example, the queue can become this:

```
template<class T, int size>
class queue {
  T *q[size];
  int in, out; // indexes
public:
  queue() : in(0), out(0) {}
  void add(T * i);
  T* get();
};
```

The second template argument, size, is an int which is used directly in the class, replacing the previous enumeration. Notice also that the definitions for add() and get() are no longer inline. An out-of-line member function definition for a template must follow a certain form; the template declaration comes first, and the class name must be followed by its template argument list, like this:

```
template<class T, int size>
void queue<T, size>::add(T * i) {
  q[in++] = i;
  if(in >= size) in = 0; // wrap
}

template<class T, int size>
T* queue<T, size>::get() {
  if(out == in) return 0;
  T *result = q[out++];
  if(out >= size) out = 0;
  return result;
}
```

This information is required so the compiler can match the definition to the declaration in the class, and also so it can generate the right information to use when creating definitions. The compiler must generate different member function definitions for each template variation you use. These variations are completely distinct, even down to the size parameter. This means that if you create classes which differ only by an int for the same template, like this:

```
queue<float, 10> fq1, fq2;
queue<float, 11> fq3;
```

You can assign fq1 = fq2 because they are the same type, but you can't say fq3 = fq1 because they are different types. Note that there is no operator=() defined for queue, so the compiler creates one for you in the first case to assign between objects of the identical type. However, it can't create one in the second case because the objects are of different types (even if we happen to know they are quite similar). It makes sense because the objects are different sizes — what would the assignment mean in such a case? This isn't something you'd want the compiler to guess at; you have to take control by writing your own operator= if you in fact want to assign between these two types.

Template Implementation Files

In the download file QUE3.CPP the definitions for queue<T, size> are not inline, but they still appear before they are used in queue<float, 10> and queue<float, 11>. The compiler must know how to generate all the member functions for all the variations of the queue class. As long as the member functions are all inline or they appear in the same compilation unit where queue objects are created, there's no problem.

But there is a problem if you want to create a header file with the template class declarations, and a separate implementation file containing the template member function definitions. This is

what you normally do with ordinary classes, so it makes sense to do it with templates. The problem is that when you use separate compilation, the compiler doesn't have the template member function definitions available in the file where you're creating all the various kinds of templates (different queues, for example). Thus, it cannot generate the necessary member functions, and you'll get a link error saying that those functions can't be found.

There are two solutions to this problem. The first (and the best, for the time being) is simply to place all template member functions in the header file and eliminate the separate implementation file. The second approach is to leave the template member functions in a separate implementation file and use two new switches which control template creation. The first, *#pragma option -Jgx*, should go at the end of the header file containing the template class declaration. It tells the compiler to generate external references to the template member functions, rather than trying to generate them in this file.

The second pragma, *#pragma option -Jgd*, should be at the beginning of the template member function definition file, after the header file has been included (so it changes the state set by the pragma in the header file). The switch tells the compiler to create public definitions for template instances. However, you must do one more thing. No member functions will be created unless the compiler has a template declaration with the appropriate arguments.

Therefore, somewhere in the file (I put mine at the beginning), you must include the header files for all the types you plan to use, and then effectively make "dummy" type declarations.

You can see an example of separate compilation with templates in the DynArray example (adapted from [1]) in the download files. In it, the dummy type declaration occurs in the file DYNARRAY.CPP:

```
typedef DynArray<String>
    dummy_type_string_vector;
```

This typedef name is never intended to be used, so it should be as unlikely as possible. The sole reason it's there is to tell the compiler to generate the function definitions for the array.

As you can see, using separate compilation for templates is not particularly convenient, so you may want to consider sticking with the inline approach. In the future, we can expect improvements to make separate compilation easier to use with templates.

Containers & Iterators

In a simple array, you can easily have more than one index. So far, the containers we've seen have controlled access to themselves. Sometimes this makes sense. However, it can be inconvenient to be stuck with a single "cursor" which is bound inside the container. Suppose you want to look at more than one spot in the container at one time? Tying containment of objects together with traversal of a container is generally limiting. To get around this, we can use a "control abstraction" (an abstraction representing control separately from containment). One type of control abstraction is an iterator.

> **In C++ 3.0, there are two types of operator++, a pre-increment, indicated by arguments, and a post-increment, indicated by a single int argument.**

Consider a very simple array type:

```
template<class T, int size>
class array {
  T a[size];
public:
  T& operator[](int i) {
    // for assignment
    if(i >= size || i < 0) {
      cerr << "array exceeded"
           << endl;
      return *new T;
      // dummy value
    }
    return a[i];
  }
  friend class arrayIter<T, size>;
};
```

Notice that this array holds, not pointers to objects, but the objects themselves. This is something that templates handle quite easily, but it puts certain restrictions on the objects which are to be held in the container. For instance, the objects must have a default constructor (because the array 'a' must be made without arguments) and a copy-constructor and operator= if the compiler-generated versions are insufficient (for more information, see [1]). You won't know if everything works okay until you actually try to make an instance of the array for a particular class, since the compiler can't check your template until it's expanded.

The operator[ ] returns a reference to an object. When using pointers, you can simply return 0 if there's no object at a location or if you run off the end of the collection. References are different — you must always return a reference to a real object. In this class, if an error occurs, I log a message and return a reference to an object created on the heap. Normally this would be a very bad practice because the user doesn't know that the object is on the heap, and thus doesn't know to destroy it. Here, however, an error has occurred and rather than terminating the program it is sometimes better to continue limping along, since that way you can collect more information about the problem.

The operator[ ] can be used to assign values to the array elements (since it returns a reference, it can be used as an lvalue, on the left-hand side of the equal sign). It can also fetch values from the array. But instead, consider a separate abstraction which allows you to index through the array:

```
template<class T, int size>
class arrayIter {
  static T dummy;
  array<T,size>& ar;
  int index;
public:
  arrayIter(array<T, size>& A)
    : ar(A), index(0) {}
  T operator++() {
    // pre-increment
    if(++index >= size)
      return dummy;
    return ar.a[index]; // a copy
  }
  T operator++(int) {
    // post-increment
    if(index >= size) return dummy;
    return ar.a[index++]; // a copy
  }
  operator T() {
    if(index >= size) return dummy;
    return ar.a[index]; // a copy
  }
  int end() { return index >= size; }
};
```

Here, each arrayIter object has its own index

into an array. The operator T() performs an automatic type conversion, so anywhere the iterator is used the compiler can automatically convert it to type T (a String, for example) by returning a copy of the object in that location.

In C++ 3.0, there are two types of operator++, a pre-increment, indicated by no arguments, and a post-increment, indicated by a single int argument. When the compiler sees a ++ used with your object, it calls a different version of the function depending on whether the ++ is before or after the variable. If it calls the post-increment version, it simply passes a constant integer that has no meaning.

The static class member dummy is a value to be returned if the index goes out of range (this is an alternative to the previous *new T approach shown in array). Defining a template static data member must be done just like defining any other external template member:

```
template<class T, int size>
  T arrayIter<T,size>::dummy(0);
```

Now we can create an array:

```
array<String, 5> S;
```

And fill it with String objects using the operator[] from array and the operator= from String:

```
S[0] = "are";
S[1] = "we";
S[2] = "having";
S[3] = "fun";
S[4] = "yet?";
S[5] = "too much!"; // exceeds array
```

The iterator arguments must exactly match those of the array:

```
arrayIter<String, 5> si(S);
```

Now you can use the iterator as a regular indexing tool:

```
while(!si.end())
  cout << si++ << " ";
```

Function Templates

Templates are not restricted to classes. You can use templates to create a description of a generic function which can then be applied to any type. Consider a simple function to swap the values of two variables:

```
template<class T>
void swap(T& a, T& b) {
  T tmp = a;  // copy constructor
  a = b;      // operator=
  b = tmp;
}
```

Again, the copy-constructor and operator= must have the appropriate behaviors for class T. Now we can swap any two objects. For example, the swap function can be used inside another function template, which performs a selection sort:

```
template<class T>
void sort(T L[], const int sz) {
  for(int i = 0; i < sz - 1; i++) {
    int min = i;
    for(int j = i+1; j < sz; j++)
      if(L[j] < L[min]) min = j;
    swap(L[min], L[i]);
  }
}
```

Now you can sort any array of objects, as long as it has an operator< defined. You can find a test for this class in the download file.

THE RULES FOR USING TEMPLATES WITH INHERITANCE ARE THE SAME AS FOR ORDINARY INHERITANCE, EXCEPT YOU MUST FULLY SPECIFY TEMPLATES.

Inheritance and Templates

The rules for using templates with inheritance are the same as for ordinary inheritance, except you must fully specify templates. For example, consider a class which contains the basics of a stack:

```
template<int sz>
class stackb {
  void* s[sz];
  int i;
public:
  stackb() : i(0) {}
  void push(void* v)
    { s[i++] = v; }
  void* pop()
    { return s[—i]; }
};
```

For brevity, no error checking is performed. To use this stack in a practical situation, you must tell the compiler exactly what type of pointer is being pushed and popped. To make a generic stack, we use a template with inheritance:

```
template<class T, int sz>
class stack : public stackb<sz> {
public:
  void push(T* v) {
    stackb<sz>::push(v);
  }
  T* pop() {
    return (T*)stackb<sz>::pop();
  }
};
```

All the template does is specify from void* to T*, which means there is no run-time overhead. Notice that the base class stackb must have its argument in the inheritance list.

If you create a collection class (without templates), which takes and returns void*, and use templates and inheritance to specify the class to a particular type, you have a third solution to the problem mentioned earlier concerning separate compilation with templates.

Summary

Next time you need to stash your objects somewhere, remember that all you need to do is go to the manual and find the appropriate generic collection. It's easy to use, and someone else has done all the work of design and debugging. As you can see, templates are an important way to reuse code in C++.

References

1. Eckel, B. *Using C++* 2nd ed., Osborne/McGraw-Hill 1992.

About the Author

Bruce Eckel is the author of C++ Inside & Out *(The revision of* Using C++, *Osborne/McGraw-Hill, 1992) and Borland's World of C++ video training course. He is a voting member of the ANSI C++ committee and the features editor for the* C++ Report. *He was the C++ speaker for Borland's OOP World Tours 1 & 2 and speaks regularly at Software Development and other conferences. He has a BS in applied physics, an MS in computer engineering, has been working with C++ since 1987, and owns Revolution2, a firm specializing in C++ consulting.*

turbo TECHNIX®

C++

Q I am trying to create a floating point array that is larger than 64K but my program crashes. I changed to the huge model but it didn't help. What am I doing wrong?

A You are confusing the HUGE MODEL and the HUGE KEYWORD. Unlike the FAR memory model, which defaults to far pointers for data (and code), the huge memory model *does not* default to huge pointers for data. In *any* memory model, if you wish to access all data in a data object larger than 64K you must declare it with the huge keyword. In addition, you should use farmalloc() or farcalloc() when dynamically allocating memory for such objects.

Q All of a sudden I started getting the linker error "DGROUP exceeds 64K." How do I get around this?

A All models give a program at least one 64K segment to contain GLOBAL and STATIC data. The HUGE model is unique in that it gives each source file a 64K data segment. One solution to your problem (though not necessarily the best) is to switch to the HUGE model and adjust your source so that the global and static data are divided between modules in such a way that no single module has more than 64K of static or global data.

Another solution is based on the fact that the "far" keyword forces a data object out of the default data segment into far data. By judicious use of the far keyword you can reduce the demands on the default data segment.

If you are using Borland C++ (as opposed to Turbo C++), you have a third approach; use the "automatic far data" option. This option will force any data object equal to or larger than the "threshold" value into far data, thus reducing the demand on the default data segment. This option, however, will most likely generate very inefficient code (in some cases worse than huge model), and so should be used with caution.

Finally, it is often possible (and even advisable) to resolve the problem by converting statically declared data objects into dynamically allocated objects. This will remove the object from the default data segment and take its memory requirements from the heap.

Q I changed from the small model to the large model to overcome "TEXT exceeds 64K" errors from the linker. Now I am getting "Fixup errors" from the linker. Help!

A Changing models does *not* force the project managers (IDE or MAKE) to recompile all modules. The project managers are *totally* file-date-stamp driven, and changing models does not effect the file date stamp.

If you are an IDE user, do a Build All. If you use the command line tools, force recompilation of all modules. The problem is that one or more modules are still small model and do not have "stubs" large enough to hold an address containing a segment value. When this happens, the linker reports that it can't "fixup" the external reference.

Q Does Borland C++ generate 32-bit code? I want to write programs to use the entire 8 megabytes of memory in my computer.

A No, Borland C++ currently does not support 32-bit code generation. If you are developing Microsoft Windows applications, this is not an issue since Windows supplies the memory management necessary to access your extended memory. If you are developing DOS applications, you should look into DOS Extenders. A DOS Extender will allow you to access a flat 4 gigabytes of extended memory via standard 'C' allocation functions. Both Ergo (508-535-7510) and Phar Lapp (617-661-1510) have announced versions of their DOS extenders that support Borland C++.

Turbo Pascal

Q On my dual floppy system, when I start Turbo Pascal 6.0, I get the copyright message and the disk light is on but nothing happens. What can I do?

A If you have a system equipped with 720K, 1.2M, or 1.44M floppies, copy the contents of the library diskette onto the compiler disk. Start Turbo Pascal by placing this diskette in the current drive and type "TURBO" from the DOS prompt. A separate library diskette is not necessary in this case. For 360K drive systems, place the compiler disk in drive B: and the library disk in drive A:. At DOS with A: as the current drive, type "PATH=B:". Then type "B:TURBO" to start Turbo Pascal.

Q What needs to be considered to link external assembly language routines into Turbo Pascal?

A If you have version 6.0, use BASM (Borland's Inline Assembler) wherever possible. The assembly source resides in the same files as your Pascal source and is assembled right along with the Pascal code. BASM greatly simplifies the process of maintaining Pascal code that uses assembly language.

If you have an earlier version of Turbo, or if you already have developed using a MASM compatible assembler (such as Borland's Turbo Assembler, TASM), follow these guidelines:

1. If using TASM, compile using "MODEL TPASCAL." Doing so will allow you to use simplified segment directives and manage the stack values for the RET instruction.

2. If using MASM, do not use simplified segment directives. Simplified directives generate segment names that Turbo Pascal may not recognize. Name the segment for code CODE or CSEG and the segment for data DATA or DSEG. The data segment must be uninitialized. See the Turbo Pascal manuals for specifics regarding Turbo Pascal's parameter passing conventions.

Q In my Turbo Pascal OWL program, how can I change the static text to white on black and my dialog's background color to blue?

A Create a descendant of TDialog that reacts to the message WM_CTLCOLOR. For its message record, WParam represents the display context of the window, LParamLo is the handle to the window and LParamHi represents the ctlcolor_ constant to be addressed.

For LParamHi, in this case, CtlColor_Static is sent when static text is to be drawn and CtlColor_Dlg controls the color of the dialog itself.

Here's code that will produce white on black text in a blue dialog. Note: a data field called BackBrush has been added to the definition of TTestDialog to hold the blue background brush used for the blue background. For the static text, the stock object Black_Brush and the function SetTextColor are used for the white on black text.

```
type
 TTestDialog = object(TDialog)
  BackBrush : HBrush;
  constructor TTestDialog.Init
     (AParent: PWindowsObject;
      AName: PChar);
  destructor TTestDialog.Done;
  procedure TTestDialog.WMCtlClr
     (var Msg: TMessage); virtual
     WM_FIRST + WM_CTLCOLOR;
 end;

constructor TTestDialog.Init
   (AParent: PWindowsObject;
    AName: PChar);
begin
 TDialog.Init(AParent, AName);
 BackBrush :=
   CreateSolidBrush
     (RGB(0, 0, 255));
end;

destructor TTestDialog.Done;
begin
 TDialog.Done;
 DeleteObject(BackBrush);
end;

procedure TTestDialog.WMCtlClr
   (var Msg: TMessage);
 begin
 case Msg.LParamHi of
   CtlColor_Static:
     begin
     SetTextColor(Msg.WParam,
       RGB(255, 255, 255));
     SetBkMode(Msg.WParam,
       transparent);
     Msg.Result :=
       GetStockObject
         (black_Brush);
    end;
   CtlColor_Dlg:
    begin
     SetBkMode(Msg.WParam,
       Transparent);
     Msg.Result := BackBrush;
    end;
  else
   DefWndProc(Msg);
  end;
end;
```

ObjectVision

Q Is it possible to add user-defined functions to the functions used in ObjectVision?

A Yes. One of the new functions available in ObjectVision is @REGISTER, which permits users to incorporate specialized functions written as DLLs (Dynamic Linked Library). This makes it possible to tailor the ObjectVision application for even the most demanding projects. ObjectVision also allows for DLL callbacks. External DLLs may call ObjectVision to execute commands within ObjectVision as well. Once a DLL function is registered, it is included in the Function Paste list, as if it were a normal ObjectVision function.

Q When complex decision trees from one application are needed in another application, can they be copied over or must they be re-created?

A Open the application with the items to be copied, and open the second application in a new window. Use the clipboard to copy necessary forms, trees or other objects desired between the two applications to avoid wasting precious development time.

Q Can the menus be customized?

A With the new version of ObjectVision you can decide what menu items will be available to end users. You can add or remove and change menu items to suit the needs of the users. Most menu items are now available in functions that may be placed directly on the form, so the user can select Print or Clear, for example, from a button rather than the menu.

Q Can I call other applications from within ObjectVision 2.0?

A Yes. @EXEC is a new function provided to call other applications. The other applications must be in your DOS path. In the case of DOS applications, the size at which the application opens will be determined based on your _DEFAULT.PIF file display setting (full screen, minimized or windowed).

Q Can applications be password-protected to avoid unwanted modification?

A ObjectVision 2.0 offers passwords which prevent unwanted changes from being made to an application. Users who do not know the password will be able to open the application and use it, but the Tools menu will not be available to modify the application. In addition, applications connected to password-protected database files will enforce the database passwords for data security. Finally, you can create a decision tree which checks for a particular string to be entered upon opening the application to keep unauthorized users from opening an application. Should an incorrect entry be provided, the APPEXIT command can be called to dump the user back to the Program Manager in Windows — effectively locking unwanted users out of the application.

BORLAND'S SOURCE HOTLINE

Source listings are available electronically on CompuServe, BIX, GEnie and the Borland DownLoad BBS service.

For CompuServe, type GO BPROGA and then type LIB 15. On BIX, type JOIN BORLAND/LISTINGS. For GEnie, type BORLAND, then select Software Libraries. For Borland's DL BBS, use your modem to dial (408)439-9096, then select the Borland Language Express section.

If you have access to a FAX machine, you may order a FAX copy of the source listings by contacting Borland's Technical Support Department at (408) 438-5300 and selecting the TechFax Service from the available options.

Photograph courtesy Monterey Chamber of Commerce

BORLAND International Developers Conference '92

by Christine Sherman

VISIONS: INTERACTIVE EDUCATION FOR THE '90s!

Consider the ingredients of the ideal software development conference: addresses by key industry figures from whom you want to hear, intensive training sessions for your level of ability, get-togethers where you can mingle with industry experts, in addition to tutorials, technical sessions, computer labs, and special events.

The conference theme is "Visions"; the goal is to shed light on today's technologies and take a look at future development directions.

WHERE: MONTEREY, CALIFORNIA
WHEN: APRIL 12-15, 1992

Borland has lined up exceptional speakers from all areas of the software development industry to present technical sessions on topics from design and development to operating system and software issues. Here are some highlights:

- Optional half-day tutorials on Sunday, including *Introduction to OOP, C++, Turbo Pascal for Windows, Object-Oriented Design*, and *ObjectVision.*
- Four days of intense training in OOP, C++, Pascal and Visual programming.
- Sessions with the top industry leaders and Borland developers.
- The latest Borland strategy and future directions from Philippe Kahn, Rick Schell and Gene Wang.
- News and views on the future of development, with speakers from IBM, Intel, Borland, Novell, Microsoft.
- Casino Night, Exhibits, Vendor Reception, Theme Dinner, Computer Lab and Game Room.

Here is a sample of what you can expect to walk away with at the end of the 4 days:

- You will be *well trained* in OOP, C++, Pascal, ObjectVision, DOS, Windows and OS/2.
- You will have a *proceedings binder* full of articles and presentations from the speakers.
- You will have made *new contacts* throughout the software development industry.

There will be over 60 breakout sessions organized into technical tracks: *C++ Programming, Pascal Programming, End User Programming, Management & Design* and *Systems & Software.*

SPECIAL CONTESTS

- Development Shoot-Out: Sponsored by *PC Week Magazine.* Attendees can sign up and use the tools of their choice to build an application specified by columnist Peter Coffee.
- Best Hacks: Sponsored by *PC Techniques Magazine.* Attendees can bring their best hacks (short pieces of code that accomplish amazing things). Winning hacks will be published in *PC Techniques.*

CONFERENCE-AT-A-GLANCE

Sunday, April 12, 1992
AM TUTORIALS
  Introduction to OOP
  Introduction to Object-Oriented Design
USER GROUP AND TUTORIAL LUNCHEON
COMPUTER LAB
PM TUTORIALS
  Introduction to C++
  OOP using Turbo Pascal for Windows
  ObjectWindows for C++
  App Creation with ObjectVision
VENDOR EXHIBITS
CASINO NIGHT

Monday, April 13, 1992
CONTINENTAL BREAKFAST
OPENING SESSION
  CEO Perspective - Philippe Kahn
  Borland Languages - Gene Wang
  Language Futures - Rick Schell
MORNING BREAKOUT SESSIONS
LUNCH SPEAKER: Lee Reiswig, IBM
AFTERNOON BREAKOUT SESSIONS
VENDOR RECEPTION
BIRDS OF A FEATHER SESSIONS

Tuesday, April 14, 1992
INTERNATIONAL PASSPORT BREAKFAST
CONTINENTAL BREAKFAST
MORNING BREAKOUT SESSIONS
LUNCH SPEAKER: Ron Whittier, Intel
AFTERNOON BREAKOUT SESSIONS
VENDOR EXHIBITS
THEME DINNER - *VISIONS*
  at the Monterey Bay Aquarium

Wednesday, April 15, 1992
CONTINENTAL BREAKFAST
MORNING BREAKOUT SESSIONS
MEET THE BORLAND TEAMS
VENDOR EXHIBITS
VENDOR LUNCH
FINAL SESSION

Call now to register and receive your FREE Borland language product: Turbo Pascal Professional, Turbo Pascal for Windows, Borland C++ & Application Frameworks, ObjectVision, or the *Visions* Video Library.

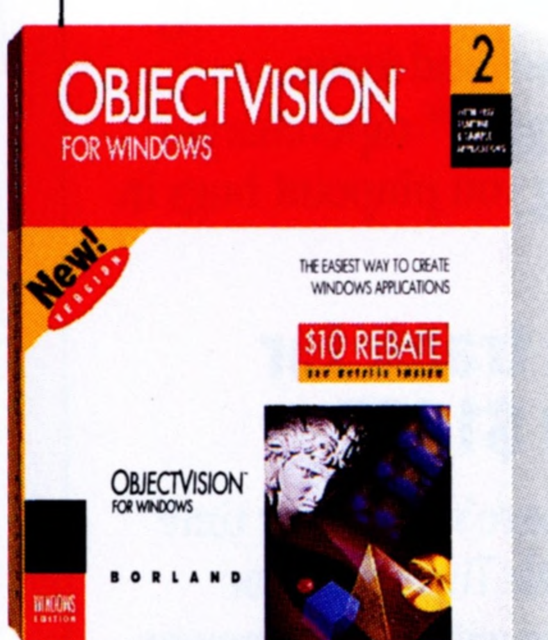
CONTACT:
CT Meeting Planners, Inc.
Borland Developers Conference
731 Main St., Suite C3
Monroe, CT 06468
1-800-942-TURBO (USA & CANADA)
1-203-452-5388 (Connecticut)
1-203-261-6227 (International)
FAX: 1-203-261-3884

Get your creative license to Windows

Everyone wants custom Windows applications. But delivering them is easier said than done. Until now. With new ObjectVision™ 2.0 from Borland you can create full-fledged Windows applications quickly *and* easily. Now you can get creative.

Visual programming makes it easy

Using ObjectVision 2.0, you create complete Windows applications visually.



Simply **draw** the application interface with the WYSIWYG forms tool, **add** the program's underlying logic in the form of graphical decision trees (much like a spreadsheet), then **connect** to your data. You can do it all just by pointing and clicking. No programming language is needed! Here's what you'll use:

- **ObjectBar™ and Property Inspector:** Add objects and set their properties with a simple tap of the mouse button.
- **Incremental Development:** See the results of your work instantly. There's no waiting through a compile/debug step.
- **Event Trees:** Trigger any action through an *event* such as clicking the mouse or closing a form. Event trees can be attached to any object for full control of the application's behavior, including custom menus, external commands, data manipulation and more.
- **More than 150 spreadsheet-style @functions.**

And you get **free** runtime and password protection! So now you can distribute applications at no cost and, if necessary, password-protect them to prevent modifications.

With power to spare!

ObjectVision comes with an incredible array of features designed to simplify your Windows development:

- **DLLs:** Connect to standard **DLLs** or create your own in C, C++ or Pascal. You can even execute callbacks and ObjectVision functions in DLLs.
- **Filters:** Can be attached to links letting you create sophisticated form-based queries.
- **OLE client:** Will link to any OLE server.
- **SQL connection** (available separately): for Sybase, SQL Server and MDI gateway to DB2.
- **Intelligent Prompting™:** Asks the user for necessary information at the *right* time and in the *right* sequence. Decisions are based on value tree logic.
- **Integral data connections:** Link ObjectVision fields to database fields found in Paradox,® dBASE,® Btrieve and ASCII formats. Just point and click. You can even create new database files in these formats!

Plus you get unlimited flexibility with one-to-many database links, virtual fields, cascading delete/updates and a host of other options.

Make it EZ on yourself—order now!

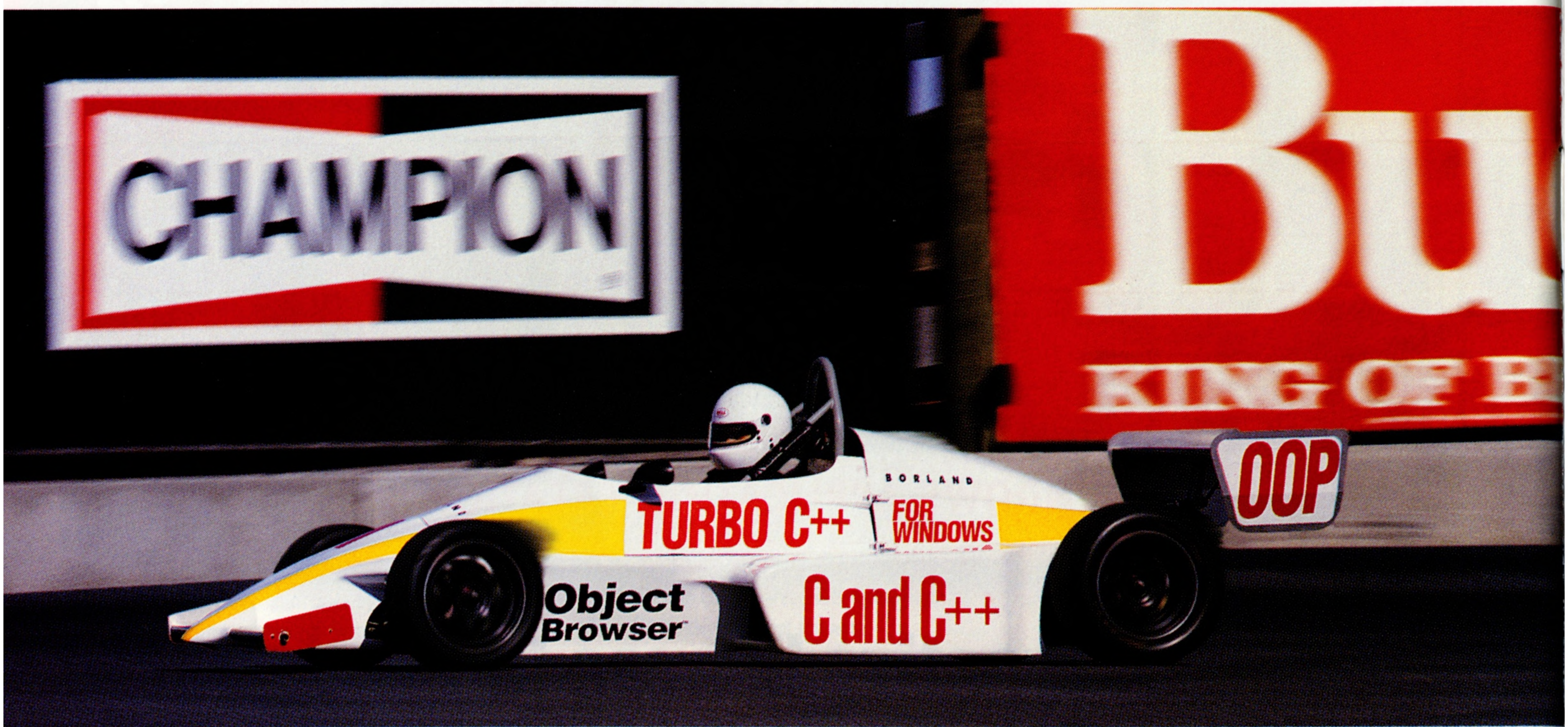
New ObjectVision 2.0 makes creating Windows so fast and easy, you've got to try it to believe it.

To order, use the attached order form or call 1-800-331-0877 or see your retail dealer.

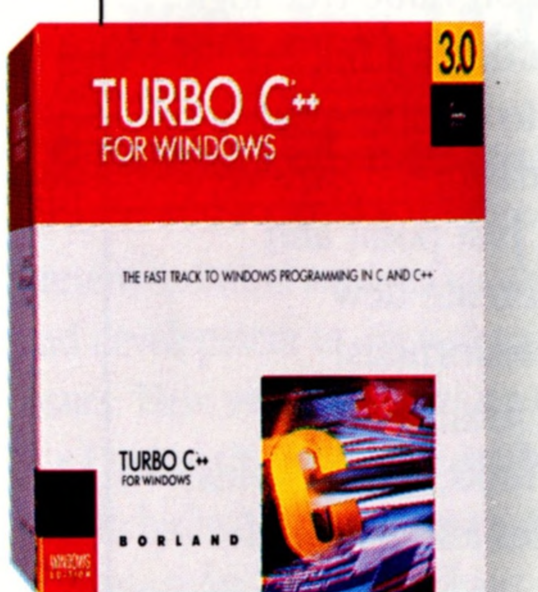
B O R L A N D

The Leader in Object-Oriented Programming

Introducing Borland's Turbo C++ for Windows



The fast track to Windows development



With Turbo C++ for Windows you develop electrifying Windows applications *fast*, even if you've never programmed in Windows before! Because Turbo C++ for Windows comes with the advantage of Borland's superior Object-Oriented Programming (OOP). Advanced technology the competition can't match.

The right equipment for your race to Windows

Turbo C++ for Windows makes Windows applications development fast, easy and *fun*. You get the **ObjectWindows**™ application framework, a ready-made user interface for Windows that you can simply plug into your application or customize any way you want. Now you can start programming in Windows immediately without having to know the intricacies of the Windows API!

Features that leave the competition in the dust

Turbo C++ for Windows comes complete with an amazing array of powerful, easy-to-use features:

The **Windows-hosted IDE** lets you edit, compile and run Windows applications all from within Windows, resulting in incredible productivity.

The new **ObjectBrowser**™ tool shows you the relationships between objects, so you can easily navigate through your code.

Don't get rid of your standard DOS programs! Simply run them under Windows by recompiling with the **EasyWin**™ library.

The new **SpeedBar**™ tool makes Windows development amazingly intuitive and fast, employing innovative icons to represent frequently used menu items. You have to see it to believe it!

In addition, you get state-of-the-art productivity tools:

Resource Workshop lets you create slick, sophisticated Windows user interface components, including menus, dialog boxes and icons.

Turbo Debugger® supercharges debugging, letting you pinpoint bugs in record time.

Get on track for only 149^{95}

At only 149^{95},* there's no better time to get on track with Turbo C++ for Windows. Because with all that power for so little money, the faster you order, the faster you'll get up to speed.

ORDER NOW!
To order, use the attached order form or call 1-800-331-0877 or see your retail dealer.

BORLAND

The Leader in Object-Oriented Programming

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS MAIL PERMIT NO. 200 SANTA CRUZ, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

BORLAND
TECHNICAL DEVELOPMENT CENTER
1800 Green Hills Road
P O Box 660005
Scotts Valley CA 95067-9985

Borland Ranked #1

"Best Application Software in Customer Satisfaction in Small to Medium-Sized and Large Businesses." —J.D. POWER AND ASSOCIATES

Best software publishers in terms of customer satisfaction among small, medium and large sized business end users. Indicators include software capability, ease of use and customer support.

The Borland Advantage

- Toll-free ordering: 1-800-331-0877.
- 24 hour - 7 day ordering by FAX: 1-408-439-9119.
- All products come with our 60-day money back offer — *No Questions Asked.*
- Unlimited technical support.

- ❑ *Yes!* Add my name to receive the next complimentary issue of Borland Language Express!
- ❑ *Yes!* Enter my name in the *Borland Language Express Sweepstakes!*
- ❑ I am also interested in the '92 BORLAND INTERNATIONAL DEVELOPERS CONFERENCE.

Name: ______________________________

Title: ______________________________

Company: ______________________________

Address: ______________________________

City: ______________ State: ______ Zip: __________

Daytime Phone Number: (   ) ______________________

Are you currently using Borland products? ❑ Yes ❑ No

If yes, please indicate which products:

❑ Turbo C++	❑ Turbo Pascal for Windows	
❑ Turbo C++ for Windows	❑ Borland C++	❑ ObjectVision
❑ Turbo Pascal	❑ Borland C++ & Application Frameworks	❑ Video training courses

1. Which operating system(s) are you currently using:

   ❑ DOS ❑ Windows ❑ Mac ❑ OS/2 ❑ UNIX

2 Which operating system(s) are you planning to use in the future:

   ❑ DOS ❑ Windows ❑ Mac ❑ OS/2 ❑ UNIX

Do A Friend A Favor.

Give this card to a friend or co-worker — and you'll honor them with a FREE issue of *Borland Language Express!* Then get ready to discuss the latest innovations and techniques showcased in future issues.

IT'S FREE!

BORLAND
1800 Green Hills Road
P.O. Box 660005
Scotts Valley, CA 95067

BULK RATE
U.S. POSTAGE
PAID
Borland International, Inc.